

|  |     |
|--|-----|
| 1. Home  | 4   |
| 1.1 CodeSmith Generator API                              | 4   |
| 1.1.1 Using the Generator SDK                            | 5   |
| 1.2 User's Guide   | 6   |
| 1.2.1 Welcome to CodeSmith Generator                     | 6   |
| 1.2.2 Installing and Upgrading                           | 7   |
| 1.2.2.1 Installing CodeSmith Generator                   | 7   |
| 1.2.2.1.1 Changing CodeSmith Generator                   | 11  |
| 1.2.2.2 Uninstalling CodeSmith Generator                 | 13  |
| 1.2.2.3 Upgrading CodeSmith Generator Templates          | 14  |
| 1.2.2.4 Upgrading existing Property Set Xml Settings     | 17  |
| 1.2.3 Introduction and Tutorials                         | 17  |
| 1.2.3.1 Main Features                                    | 17  |
| 1.2.3.2 What's New                                       | 18  |
| 1.2.3.3 Tutorials  | 22  |
| 1.2.3.3.1 Getting Started                                | 22  |
| 1.2.3.3.2 Writing Your First Template                    | 28  |
| 1.2.3.3.3 Write a Template with Database Metadata        | 33  |
| 1.2.4 Visual Studio Integration                          | 45  |
| 1.2.5 Using Template Explorer                            | 49  |
| 1.2.5.1 What is Template Explorer?                       | 49  |
| 1.2.5.2 The Template Explorer Toolbar                    | 49  |
| 1.2.5.3 Managing the Folder Tree                         | 49  |
| 1.2.5.4 Editing Templates                                | 51  |
| 1.2.5.5 Executing Templates                              | 51  |
| 1.2.5.6 Working with the Output Window                   | 53  |
| 1.2.6 Using the Template Editor                          | 53  |
| 1.2.6.1 Template Editor User Interface                   | 54  |
| 1.2.6.1.1 Template Editor Toolbar                        | 55  |
| 1.2.6.1.2 Template Documents                             | 56  |
| 1.2.6.1.3 The Properties Window                          | 57  |
| 1.2.6.1.4 The Output Window                              | 59  |
| 1.2.6.1.5 The Error Window                               | 59  |
| 1.2.6.2 Template Editor Features                         | 60  |
| 1.2.6.2.1 Bracket Highlighting                           | 60  |
| 1.2.6.2.2 Documentation Comment Editing                  | 61  |
| 1.2.6.2.3 Find and Replace                               | 61  |
| 1.2.6.2.4 Incremental Search                             | 63  |
| 1.2.6.2.5 Keyboard Shortcuts                             | 63  |
| 1.2.6.2.6 Line Modification Markers                      | 66  |
| 1.2.6.2.7 Outlining                                      | 66  |
| 1.2.6.2.8 Statement Completion                           | 68  |
| 1.2.6.2.9 Tab Groups and Split Windows                   | 70  |
| 1.2.6.2.10 Template Navigation                           | 72  |
| 1.2.6.2.11 Themes and Syntax Highlighting                | 73  |
| 1.2.6.3 Building, Running, and Compiling Templates       | 74  |
| 1.2.6.4 Customizing CodeSmith Generator                  | 74  |
| 1.2.7 Using Schema Explorer                              | 75  |
| 1.2.7.1 Managing Extended Properties                     | 78  |
| 1.2.8 Using the Map Editor                               | 80  |
| 1.2.8.1 Developing using a Generator Map                 | 81  |
| 1.2.9 Using CodeSmith Generator Projects                 | 85  |
| 1.2.9.1 Manage Outputs                                   | 86  |
| 1.2.9.1.1 Project Options                                | 89  |
| 1.2.9.2 Using a Generator Project inside Visual Studio   | 92  |
| 1.2.9.3 Using Generator Project from Windows Explorer    | 97  |
| 1.2.9.4 Using a Generator Project from MSBuild           | 98  |
| 1.2.9.5 Using a Generator Project from Command-Line      | 100 |
| 1.2.9.6 Anatomy of a Project File                        | 101 |
| 1.2.10 Using the Console Application                     | 103 |
| 1.2.10.1 Incorporating Generator into Your Build Process | 103 |
| 1.2.10.2 Basic Console Application Usage                 | 103 |
| 1.2.10.3 Handling Input                                  | 104 |
| 1.2.10.3.1 Specifying Properties on the Command Line     | 104 |
| 1.2.10.4 Handling Output                                 | 104 |
| 1.2.10.4.1 Default Output Files in Templates             | 104 |
| 1.2.11 Using ActiveSnippets                              | 104 |
| 1.2.11.1 ActiveSnippet Configuration                     | 108 |
| 1.2.12 Basic Template Syntax                             | 111 |
| 1.2.12.1 The CodeTemplate Directive                      | 111 |
| 1.2.12.2 Including Comments                              | 113 |
| 1.2.12.3 Declaring and Using Properties                  | 113 |
| 1.2.12.3.1 Property Directive                            | 114 |
| 1.2.12.3.2 Declaring an Enumerated Property              | 115 |
| 1.2.12.3.3 Property Validation                           | 116 |

|   |     |
|---|-----|
| 1.2.12.4 Escaping ASP.NET Tags  | 117 |
| 1.2.12.5 The CodeSmith Generator Objects                              | 117 |
| 1.2.12.5.1 The CodeTemplate Object                                    | 117 |
| 1.2.12.5.2 The Progress Object  | 121 |
| 1.2.12.5.3 The CodeTemplateInfo Object                                | 122 |
| 1.2.13 Advanced Template Syntax                                       | 123 |
| 1.2.13.1 Understanding CodeSmith Generator's Code Behind Model        | 123 |
| 1.2.13.2 Referencing Assemblies                                       | 125 |
| 1.2.13.3 Importing Namespaces   | 126 |
| 1.2.13.4 Including External Files                                     | 126 |
| 1.2.13.5 Sharing Common Code  | 126 |
| 1.2.13.6 Debugging Templates  | 126 |
| 1.2.13.6.1 Outputting Trace and Debug Information                     | 129 |
| 1.2.13.6.2 Viewing the Compiled Template Source Code                  | 130 |
| 1.2.13.7 Using Master Templates                                       | 130 |
| 1.2.13.7.1 Registering Sub-Templates                                  | 130 |
| 1.2.13.7.2 Merging Properties into the Parent Template                | 131 |
| 1.2.13.7.3 Copying Properties from the Parent Template                | 131 |
| 1.2.13.7.4 Setting Properties in a Sub-Template                       | 131 |
| 1.2.13.7.5 Rendering a Sub-Template                                   | 132 |
| 1.2.13.7.6 A Sub-Template Example                                     | 132 |
| 1.2.13.8 Writing to Multiple Outputs                                  | 134 |
| 1.2.14 Driving Templates with Metadata                                | 135 |
| 1.2.14.1 Using .NET Types   | 135 |
| 1.2.14.2 Using SchemaExplorer   | 135 |
| 1.2.14.2.1 The SchemaExplorer Object Model                            | 137 |
| 1.2.14.2.2 Connection Strings   | 138 |
| 1.2.14.2.3 Choosing Objects   | 138 |
| 1.2.14.2.4 Sorting Collections  | 142 |
| 1.2.14.2.5 Using Extended Properties                                  | 142 |
| 1.2.14.3 XML Support  | 143 |
| 1.2.14.3.1 XML Property Examples                                      | 145 |
| 1.2.14.4 Custom Metadata Sources                                      | 147 |
| 1.2.14.4.1 Adding Designer Support                                    | 147 |
| 1.2.14.4.2 Adding Property Set Support                                | 149 |
| 1.2.14.5 Generating from Source Code                                  | 149 |
| 1.2.15 Advanced Topics  | 150 |
| 1.2.15.1 Auto Executing Generated SQL Scripts                         | 150 |
| 1.2.15.2 Merge Strategies   | 151 |
| 1.2.15.2.1 InsertClass Merge Strategy                                 | 151 |
| 1.2.15.2.2 InsertRegion Merge Strategy                                | 153 |
| 1.2.15.2.3 PreserveRegions Merge Strategy                             | 154 |
| 1.2.15.2.4 Defining Your Own Merge Strategy                           | 156 |
| 1.2.15.3 Active vs. Passive Generation                                | 156 |
| 1.2.15.3.1 Using Inheritance to Enable Active Generation              | 157 |
| 1.2.15.3.2 Using Merge Strategies to Enable Active Generation         | 158 |
| 1.2.15.3.3 Using Partial Classes to Enable Active Generation          | 159 |
| 1.2.15.4 Template Caching   | 159 |
| 1.2.15.5 Version Control Support                                      | 160 |
| 1.2.15.6 Building a Custom Schema Provider for SchemaExplorer         | 160 |
| 1.2.15.6.1 Creating a Custom Schema Provider                          | 161 |
| 1.2.15.6.2 Building a Custom Schema Provider                          | 167 |
| 1.2.15.6.3 Debugging a Custom Schema Provider                         | 168 |
| 1.2.15.6.4 Deploying a Custom Schema Provider                         | 169 |
| 1.2.15.6.5 Upgrading a Custom Schema Provider                         | 169 |
| 1.2.15.7 Using CodeSmith.CustomProperties                             | 169 |
| 1.2.15.7.1 FileNameEditor   | 170 |
| 1.2.15.7.2 StringCollection   | 172 |
| 1.2.15.8 CodeSmith.BaseTemplates                                      | 174 |
| 1.2.15.8.1 OutputFileCodeTemplate                                     | 174 |
| 1.2.15.8.2 SqlCodeTemplate  | 174 |
| 1.2.15.8.3 StringUtil   | 175 |
| 1.2.15.8.4 ScriptUtility  | 176 |
| 1.2.15.9 Building a custom UITypeEditor                               | 176 |
| 1.2.15.10 Setting up a DataDirectory for Generator Connection Strings | 179 |
| 1.2.16 Frequently Asked Questions                                     | 180 |
| 1.2.17 Tips and Tricks  | 181 |
| 1.2.18 Internet Links   | 181 |
| 1.2.19 Reference  | 182 |
| 1.2.19.1 System Requirements  | 182 |
| 1.2.19.2 CodeSmith Generator Samples                                  | 182 |
| 1.2.20 Licensing and Distribution                                     | 183 |
| 1.2.20.1 Copyrights and Trademarks                                    | 183 |
| 1.2.20.2 Software Licenses  | 183 |
| 1.2.20.3 Premier Support  | 185 |

|  |     |
|--|-----|
| 1.2.20.4 CodeSmith Generator Editions .....        | 185 |
| 1.2.20.5 Product Activation and Deactivation ..... | 186 |

# Home

## CodeSmith Generator Documentation

Welcome to the CodeSmith Generator Documentation portal. All of our documentation is provided in a friendly wiki format. Please use the navigation tree on the left to locate the subject that you are interested in.

If you can not find what you are looking for, please feel free to contact us via our [Contact Us page](#).

### User's Guide

The [CodeSmith Generator User's Guide](#) is for anyone who uses CodeSmith Generator. Are you new to CodeSmith Generator? You can start by exploring the [Introductions and Tutorials](#) sections which will show you how to create custom templates and how to use CodeSmith Generator. The User's Guide is also broken down into different categories with helpful pictures and video tutorials. These rich resources will help you get acquainted with any area of CodeSmith Generator like [Template Syntax](#), [Visual Studio Integration](#), [CodeSmith Explorer](#) and much more.

### Upgrade Guide

The [CodeSmith Generator Upgrade Guide](#) is for people who are upgrading their copy of CodeSmith Generator. Just start by reading the [latest Release Notes](#) and the steps needed to upgrade your existing templates. Then, [download Generator](#) and follow the [main Upgrade Guide](#).

### Developer Resources

These resources are for software developers who want to create their own templates or extend CodeSmith Generator by using the CodeSmith Generator APIs. All of the CodeSmith Generator API Documentation can be found [here](#).

## Additional Resources

[Official Site](#)

[Downloads](#)

[Online Store](#)

[Forums](#)

[Change Log](#)

[PDF Format](#)

## CodeSmith Generator API

Welcome to the CodeSmith Generator developer's API documentation. The CodeSmith Generator API documentation allows you to discover and consume the various CodeSmith Generator features programmatically. The Generator API documentation is available both in online and offline formats.

CodeSmith Generator exposes the entire operation of the CodeSmith Generator engine through an API. Through this API, you can compile templates, retrieve any errors, create instances of templates, get the generated source code, fill in template metadata, and ultimately render a template's output to a TextWriter, file, or string. This API allows you to perform many CodeSmith Generator operations from your own code in any .NET language, and lets you programmatically execute CodeSmith Generator templates from within your own code.




Remember, CodeSmith Generator is licensed on a per-developer basis. If your application depends on programmatically executing CodeSmith Generator templates, each user must have a [license](#) to use CodeSmith Generator.

## Online Version

Please click [here](#) to view the CodeSmith Generator API documentation online, all you need is a modern web browser.

## Offline Version

If you need to view the Generator API Documentation offline or on the go, you can download the offline version by clicking [here](#). You will then be prompted to save the Generator API documentation to your computer.

 After downloading the Generator API Documentation, you will need to unblock the downloaded GeneratorAPI.chm file by following these steps.


## Using the Generator SDK

After reading this document you will know how-to download, install and use the CodeSmith Generator SDK in your applications. This document will also demonstrate the most common uses of the CodeSmith Generator API:

- Compiling a template
- Retrieving compile errors
- Creating a new template instance
- Filling in template metadata
- Rendering a template

## Download

After logging into your [account](#), visit the following [the downloads section](#) to download the latest version of CodeSmith Generator.

 It is recommended that you download the Zipped Version of CodeSmith Genreator as it includes all of the assemblies that you will need to reference in your SDK application.


## Installing the license


A license key file will be emailed to you after purchase the SDK license from the [online store](#). This license file needs to meet one of the following criteria:

- Embed the license file into your application as an embedded resource in the assembly that calls the CodeSmith.Engine.
- Place the license file into the same directory as your application.

## Creating a new project

In order to use CodeSmith Generator SDK you will need to create a new .Net 4.0 or newer project. In the example below, we will be creating a new console application.

 A C# and VB.Net sample SDK project exists in your extracted samples under the following directory (Documents\CodeSmith Generator\Samples\<VERSION>\Projects\CSharp\APISample). You may need to do some minor changes like changing the ConnectionString.

 A project using an SDK license must be Strong-Named.

## Adding project references

Next, **you are required to reference** the following assemblies:

- CodeSmith.Core.dll
- CodeSmith.Engine.dll



These assemblies will be located in the zipped version of Generator that you downloaded in the previous step.

In the example below we will also need to reference the following assemblies:

- CodeSmith.BaseTemplates.dll
- SchemaExplorer.dll
- SchemaExplorer.SqlSchemaProvider.dll

## Writing the Code

Now it's time to dive in and write some generation code! We will add the following code to our console application:

```
public static void Main(string[] args)
{
    CodeTemplateCompiler compiler = new CodeTemplateCompiler("../..\..\StoredProcedures.cst");
    compiler.Compile();
    if (compiler.Errors.Count == 0)
    {
        CodeTemplate template = compiler.CreateInstance();
        DatabaseSchema database = new DatabaseSchema(new SqlSchemaProvider(),
@"Server=.;Database=PetShop;Integrated Security=True;");
        TableSchema table = database.Tables["Inventory"];
        template.SetProperty("SourceTable", table);
        template.SetProperty("IncludeDrop", false);
        template.SetProperty("InsertPrefix", "Insert");
        template.Render(Console.Out);
    }
    else
    {
        for (int i = 0; i < compiler.Errors.Count; i++)
        {
            Console.Error.WriteLine(compiler.Errors[i].ToString());
        }
    }
    Console.WriteLine("\r\nPress any key to continue.");
    Console.ReadKey();
}
```

The above code will compile and run but requires the Stored Procedures template that can be found in the APISample project that was mentioned above. If you have any SDK API questions feel free to contact [support](#).



In addition to the methods shown in this sample, you may also find the [CodeTemplate.RenderToFile](#) and [CodeTemplate.RenderToString](#) methods useful; they let you direct the output of your templates directly to a file or to a string variable.

## User's Guide

The [CodeSmith Generator User's Guide](#) is for anyone who uses CodeSmith Generator. New to CodeSmith Generator? Start by exploring the [Introductions and Tutorials](#) sections which will show you how to create custom templates and how to use CodeSmith Generator. The User's Guide is also broken down into different categories with helpful pictures and video tutorials. These rich resources will help you get acquainted with any area of CodeSmith Generator like [Template Syntax](#), [Visual Studio Integration](#), [CodeSmith Explorer](#) and much more.

## Welcome to CodeSmith Generator



# CodeSmith GENERATOR

CodeSmith Generator is a template-based code generator that can produce code for any text-based language. Whether your target language is [C#](#), [VB.NET](#), [JScript.NET](#), [T-SQL](#), [Java](#) or even [FORTRAN](#), CodeSmith Generator can help you produce higher-quality, more consistent code in less time than writing code by hand.

Generator's familiar ASP.NET-based template syntax means that you can be writing your first templates within minutes of installing the package. The advanced [Generator Template Editor](#) helps you create and test new templates in a rapid development setting. You can also join in CodeSmith Generator's active [online community](#) to download [hundreds of ready-made templates](#) for such common development tasks as building strongly-type collection classes or creating data access layers.

[CodeSmith Generator Projects](#) and [ActiveSnippets](#) are integrated within Microsoft Visual Studio to make code generation a breeze.

CodeSmith Generator also includes a [console version](#) and a [MSBuild task](#) that you can easily integrate into your automated build process, flexible strategies for merging generated code with custom code, the [SchemaExplorer API](#) for integration with relational data sources, and the ability to hook up your own custom metadata sources.

If you're new to CodeSmith Generator, [Getting Started with CodeSmith Generator](#) will show you how to begin generating code for your own projects immediately. If you're an experienced CodeSmith Generator user, [What's New](#) will point you at the major new features in this release.

## Installing and Upgrading

### Installing CodeSmith Generator

The following pages will go over how to install, change, and uninstall CodeSmith Generator.

- [Changing CodeSmith Generator](#)
- [Installing CodeSmith Generator](#)
- [Uninstalling CodeSmith Generator](#)



Please see [this guide](#) for more information on activating CodeSmith Generator.

### Upgrading CodeSmith Generator

This section will help guide you through upgrading CodeSmith Generator and your existing templates to the latest version of CodeSmith Generator.

To upgrade to the latest version of CodeSmith Generator please follow the steps below.

1. Upgrade your existing CodeSmith Generator key using the [following form](#). If you have any questions, please contact [sales](#).
2. You will receive an email containing your new key and a download link in an email.
3. Download and run the Installer for CodeSmith Generator. **This will handle the upgrading of any previous installs of CodeSmith Generator.**
4. [Launch Template Explorer](#) and [activate](#) using the key you received in your email (step 2).

#### Additional Upgrade Guides

- [Upgrading CodeSmith Generator Templates](#)
- [Upgrading existing Property Set Xml Settings](#)

## Installing CodeSmith Generator

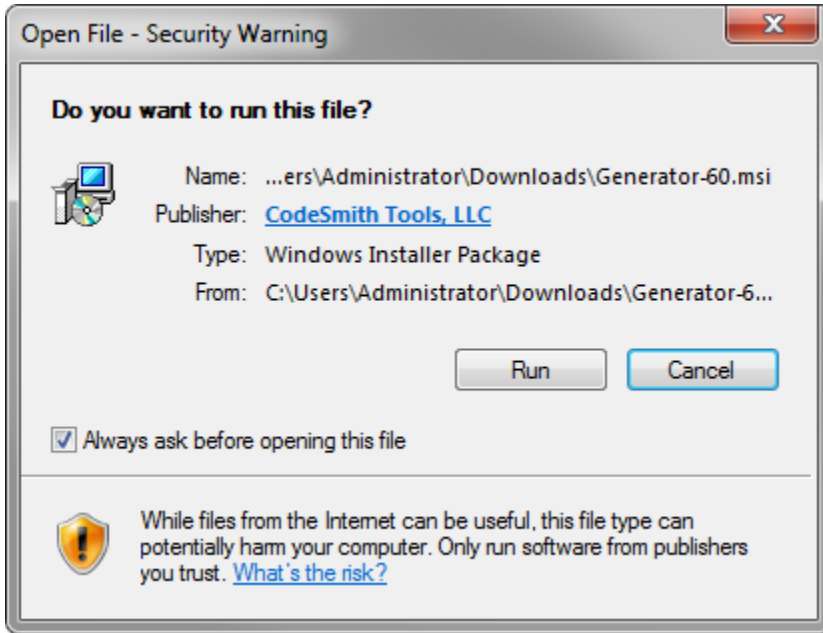
### Installing


The following section will go over how to download CodeSmith Generator, customizing the installation, and changing the samples directory.

#### *Download*

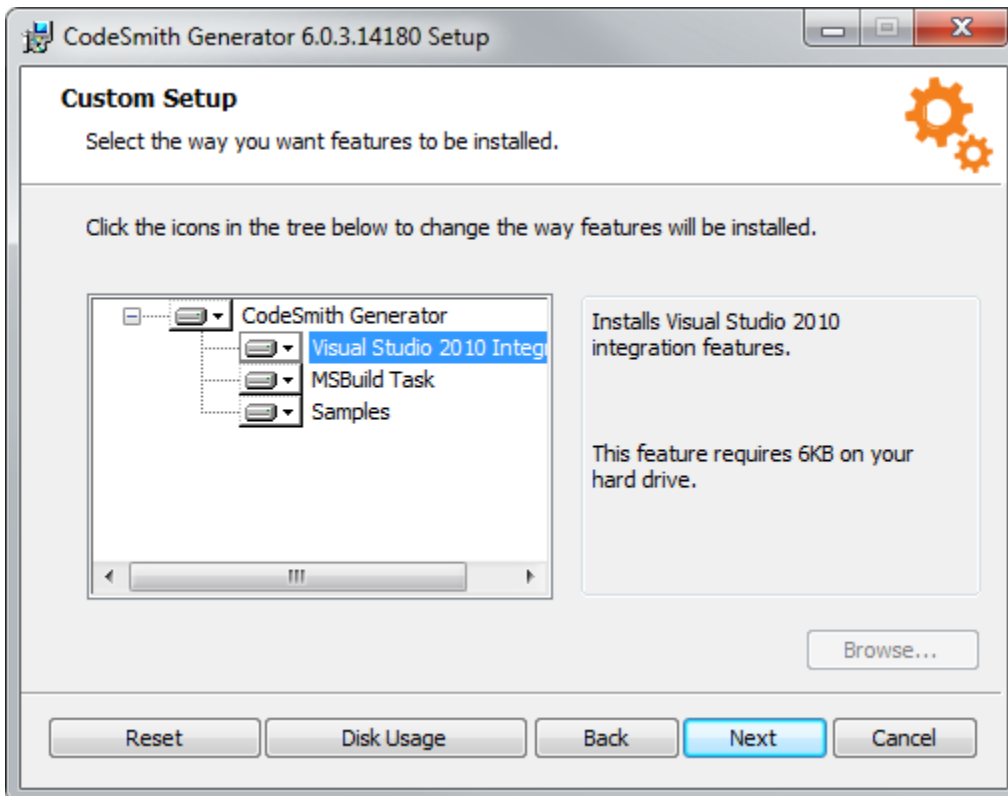
In order to download the latest CodeSmith Generator installer, just visit our [download page](#). If you need a previous version, just send us an email and we will be happy to provide you with the download.

Once you have the CodeSmith Generator installer downloaded, double click on the icon to launch it. Depending on your settings, you may be presented with this window:



 Before clicking run, please make sure that Visual Studio and previous versions of CodeSmith Generator are not running. If these programs are closed, click run to continue with the installation.

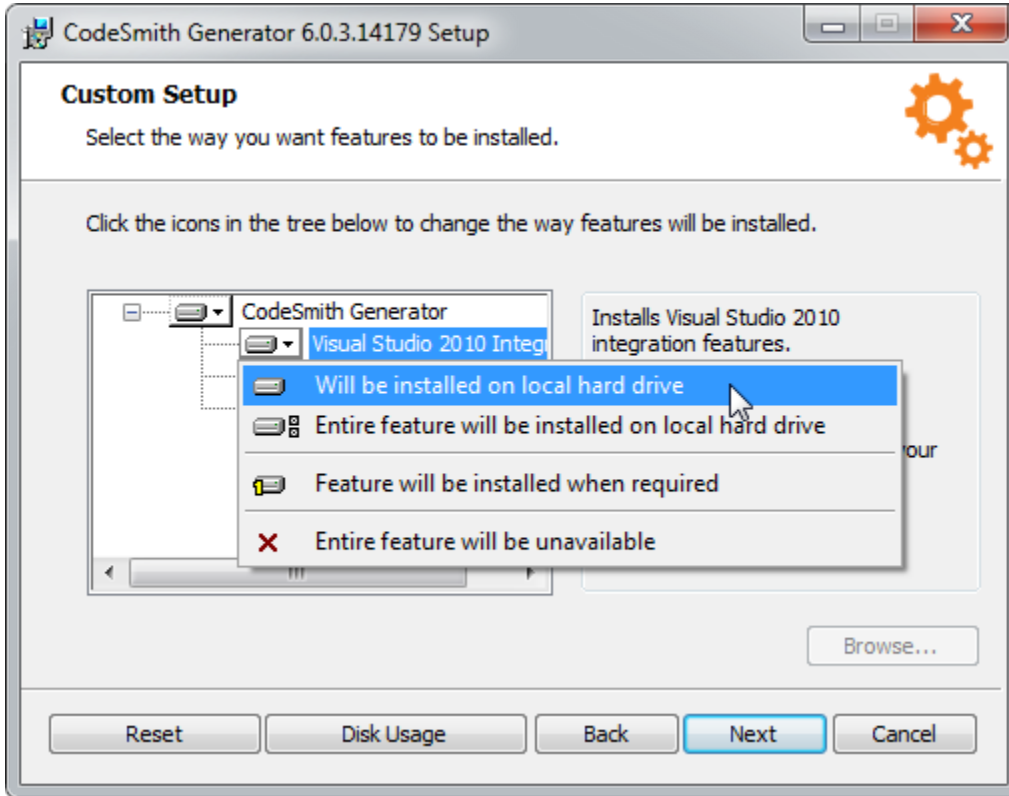
### Customizing Your Setup



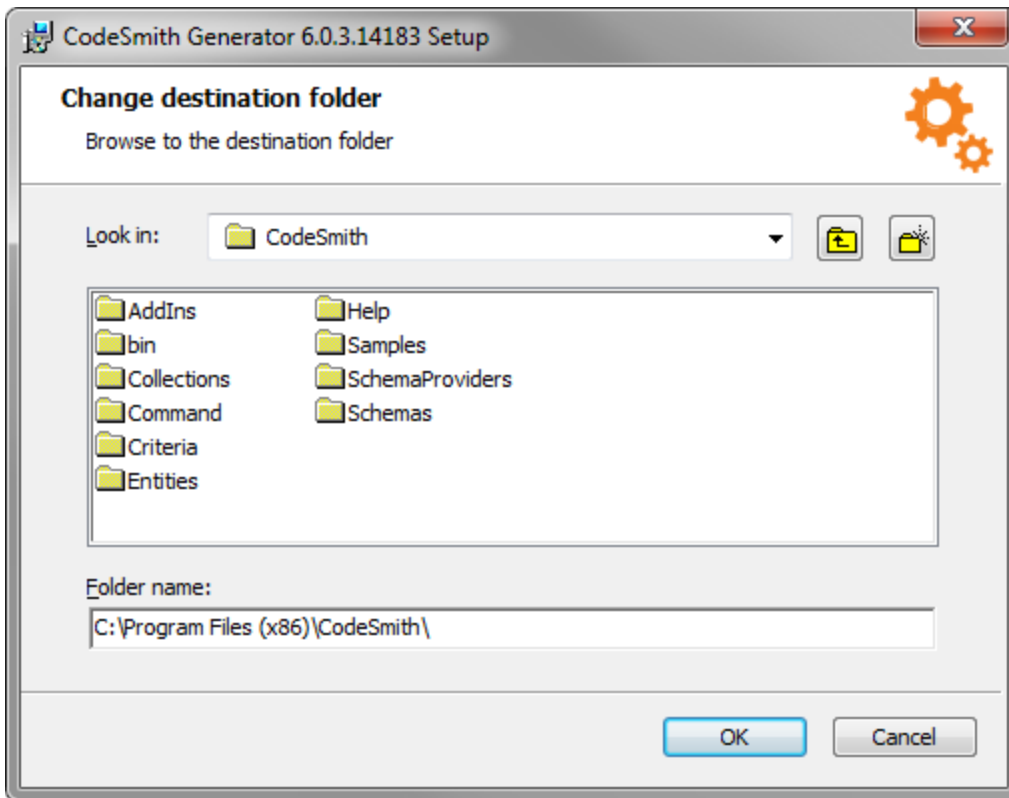


This window will allow you to customize which components of CodeSmith Generator you want to install. You can do this by clicking on the icon to the left of the component you want to change. You will be presented with the following options.

1. Will be installed on local hard drive.
  - This means the feature will be installed in your current default hard drive.
2. Entire feature will be installed on local hard drive.
  - This means that the parent and child features will be selected as "Will be installed on local hard drive".
3. Feature will be installed when required.
  - The feature will be installed when you perform an action that requires it.
4. Entire feature will be unavailable.
  - The selected feature won't be installed at all.

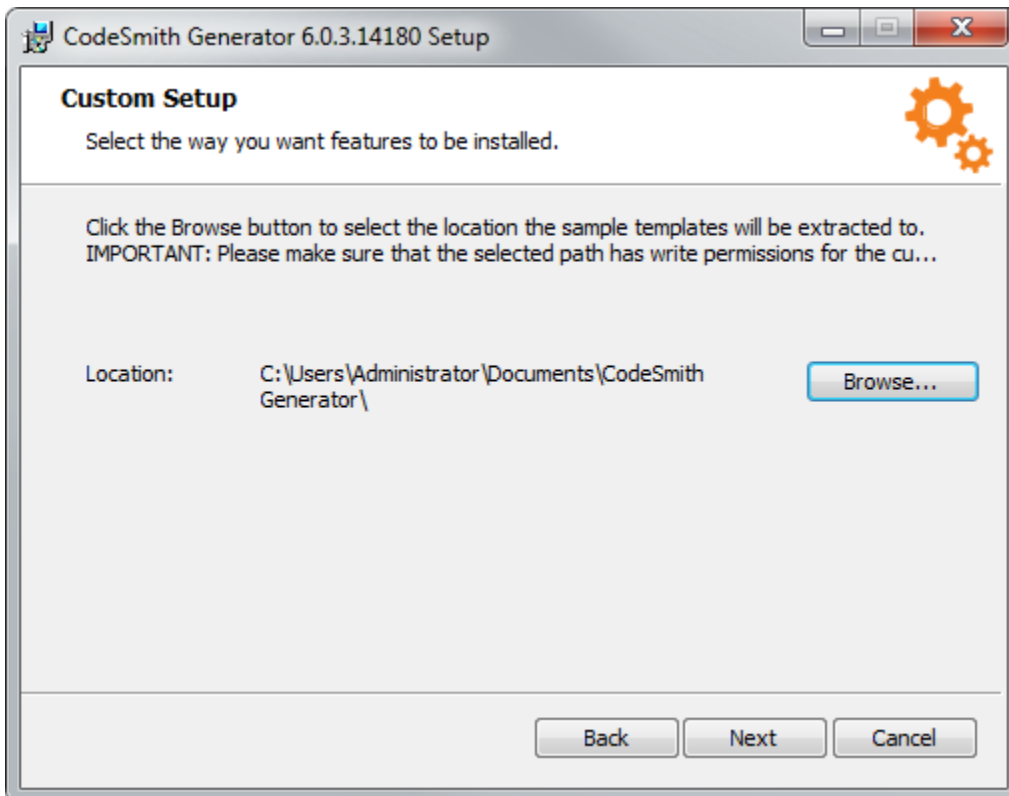


From the same window, you can also change the destination folder of CodeSmith Generator by clicking browse and selecting the file path you wish CodeSmith Generator to be placed. If you don't want to change any of these settings, just click next to continue with the default settings.

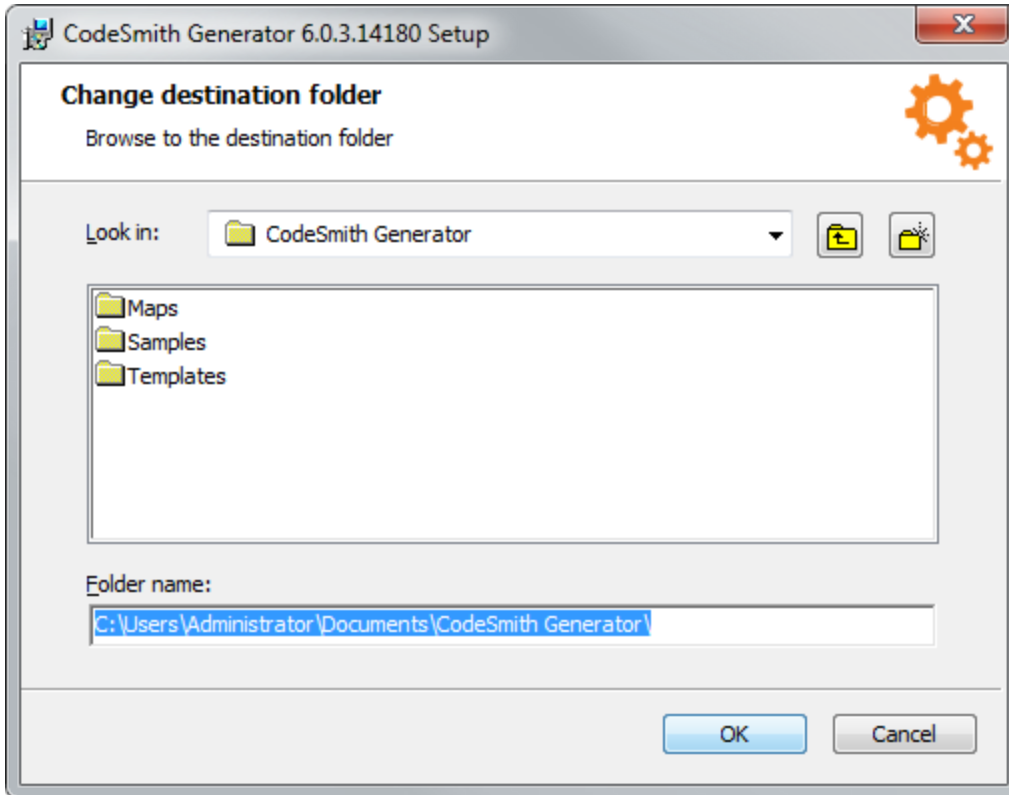


### ***Changing The Samples Directory***

After clicking next on the Custom Setup window, you will be presented with the below window.



By clicking browse in this window, you can change the file path you want your samples directory to be located in.



Please make sure that you have read/write permissions as the currently logged in user to the folder you're trying to set your samples directory to.

**Next: [Uninstalling CodeSmith Generator](#)**

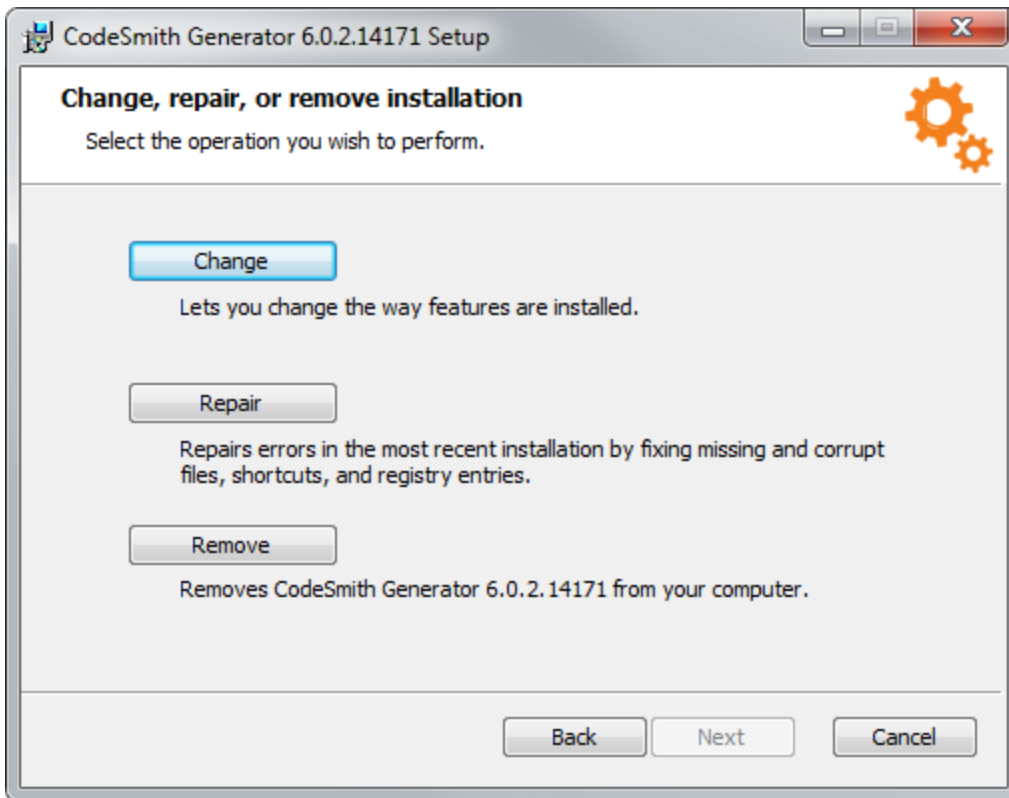
## Changing CodeSmith Generator

### Changing

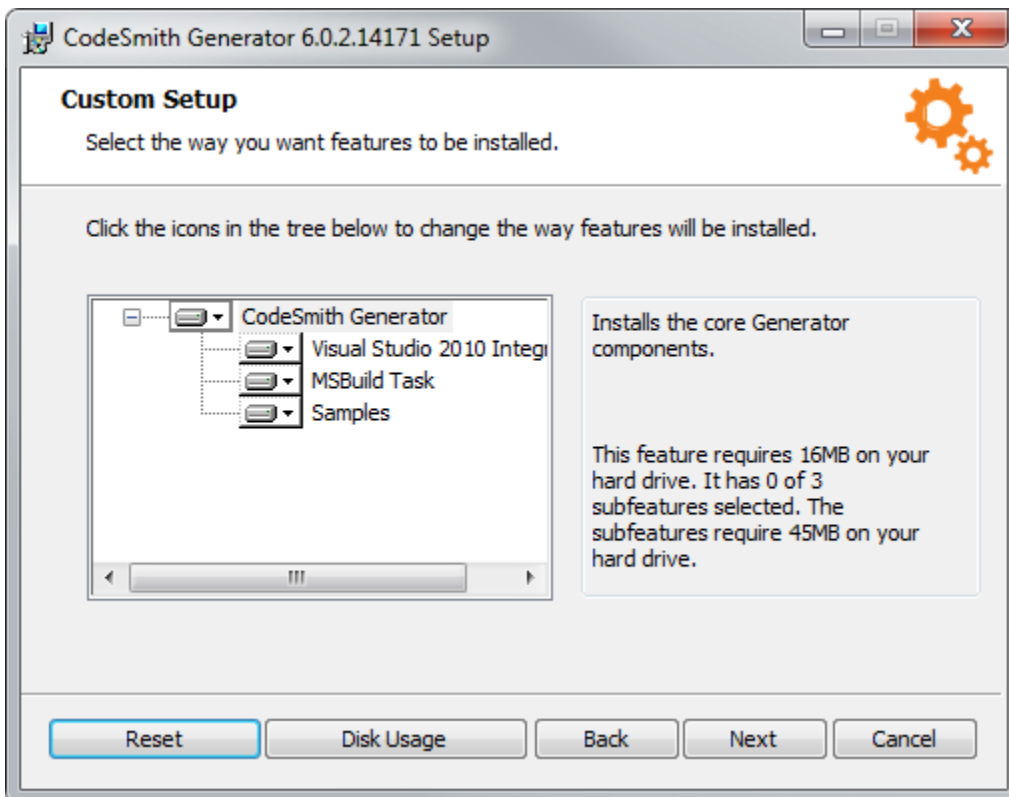
In order to change currently installed features and components of CodeSmith Generator, you will either need to:

- Go to your control panel, select Uninstall a program, find CodeSmith Generator and click 'Change' at the top bar, or right click CodeSmith Generator and click 'Change'.
- Find your original CodeSmith Generator set up file and run it. If you deleted the set up file, you can download it [here](#). If you need a previous version, please contact us via email.

Both of these options will bring you to this set up window.



Click 'Change' to continue with editing your features.



From this window, you can:

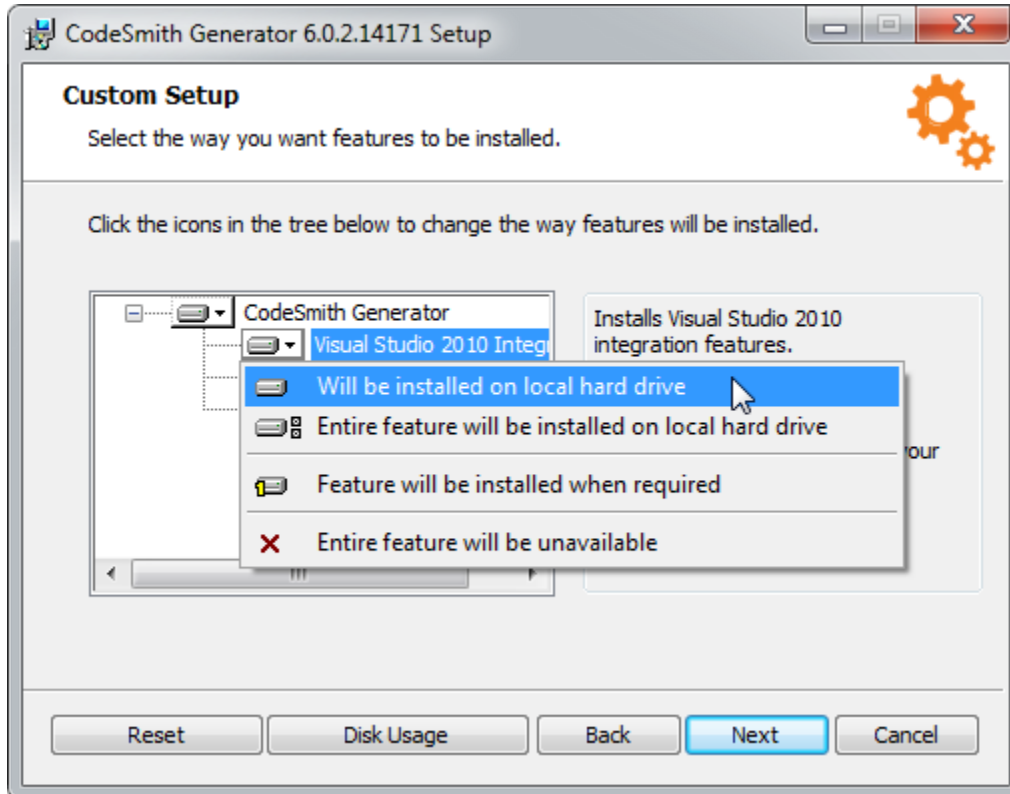
- Reset - Resets all configurations to their default settings
- Disk Usage - Displays the current space available in all of your hard drives.
- Back - Brings you to the start page.
- Next - Continues with the changes with the current changes made.

- Cancel - Cancels the setup and closes the window.

### Customizing Your Setup

The previous window will allow you to customize which components of CodeSmith Generator you want to install. You can do this by clicking on the icon to the left of the component you want to change. You will be presented with the following options.

1. Will be installed on local hard drive.
  - This means the feature will be installed in your current default hard drive.
2. Entire feature will be installed on local hard drive.
  - This means that the parent and child features will be selected as "Will be installed on local hard drive".
3. Feature will be installed when required.
  - The feature will be installed when you perform an action that requires it.
4. Entire feature will be unavailable.
  - The selected feature won't be installed at all.



Click next after you have finalized your changes to finish the setup.

**Next: Installing CodeSmith Generator**

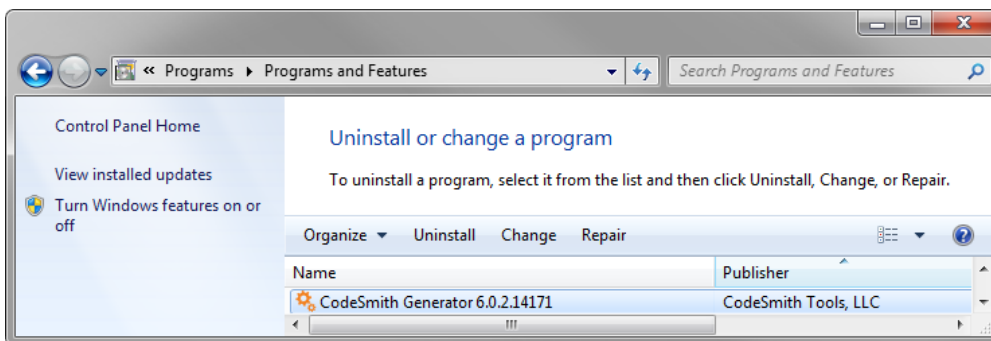
## Uninstalling CodeSmith Generator

### Uninstalling

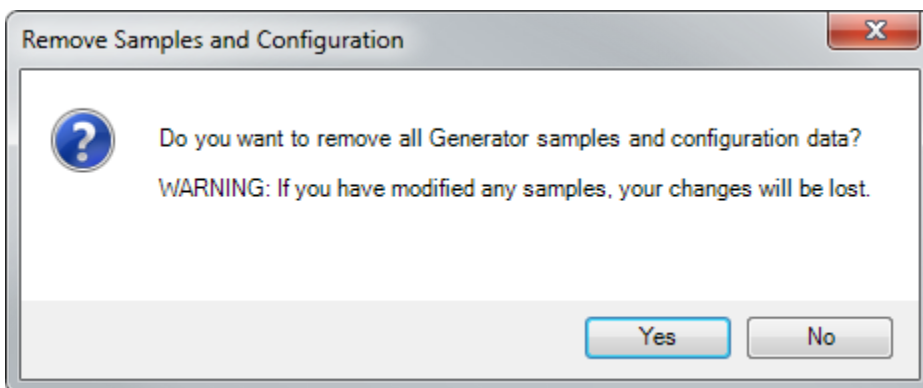
Uninstalling CodeSmith Generator is no different than most programs. Just go to your control panel, Programs, then select 'Uninstall a program'.



After you select 'Uninstall a program', the below window will appear. From there you just need to select "CodeSmith Generator (Your version number)" and then select 'Uninstall' on the top bar.



Uninstalling CodeSmith Generator is usually a one-step process, unless you already have some custom templates set up. In that case, you will be presented with this window.



Select yes to delete all samples in the samples directory you specified when you installed CodeSmith Generator. By default, the samples directory is here (C:\Users\(\User Name Here)\Documents\CodeSmith Generator\Samples), so if you need to make some back up files make them before selecting yes. Or select no to keep your samples directory for the next time you install CodeSmith Generator.

## Upgrading CodeSmith Generator Templates

We strive to ensure that CodeSmith Generator templates are 100% backwards compatible. However, in very rare circumstances break backwards compatibility to progress the platform. When we do break backwards compatibility, we ensure that the benefits of breaking compatibility greatly outway the benefits of not breaking compatibility. Please refer to this document when upgrading CodeSmith Generator to

ensure that you have the smoothest experience possible.

## Upgrading from all previous versions of CodeSmith Generator

Please read this guide when upgrading from any version of CodeSmith Generator.

### Recompiling Template Assembly References

If you have a **custom template that references an assembly that references CodeSmith Generator** or you are **using one of our Template Frameworks (E.G., PLINQO, CSLA...)** then please continue reading this step.

When upgrading from any version of CodeSmith Generator, please follow the following steps:

1. Locate your templates source code folder or source project (The source code for our [Template Frameworks](#) can be found in the templates \Source\ directory) and open the Visual Studio solution.
2. Update all assembly references that reference `CodeSmith.Engine` to `.NET 4.0` (**This does not mean that the template you write has to target .NET 4.0! This just means that your CodeSmith Generator class libraries need to be compiled as .NET 4.0**)
3. Update all of the [Visual Studio project's assembly references](#) in your Solution that reference CodeSmith assemblies. The references need to be updated to use the new CodeSmith Generator assemblies which are located in the Generator Program Files folder.
4. Add a project reference to **CodeSmith.Core** to all Visual Studio projects that reference CodeSmith assemblies. This assembly is located in the CodeSmith Generator Program Files bin folder.
5. Rebuild your Solution. **If you are updating an existing Template Framework Solution. Please ensure that each projects compiled assemblies are being copied to the correct folder after build (E.G., from the \ProjectName\bin\debug folder to the templates \common\ folder).**
6. Regenerate.



The source code for our [Template Frameworks](#) can be found in the templates Source directory.

### Downloading the latest Templates

CodeSmith Generator ships with the latest version of the templates so there is no need to go out and download the latest set of templates. However, the latest set of templates can be found on the [CodeSmith Generator Google Code project](#).

### Updating existing CodeSmith Generator Projects

After upgrading to the latest version of CodeSmith Generator, please ensure that your [Generator Project files](#) are up to date. You can do this by opening [Manage Outputs](#) and ensuring that the template location points to the latest version of the templates.

## Upgrading from CodeSmith Generator 2.x, 3.x, 4.x, 5.x

### Updates to CodeTemplates Validate method

If you are using a custom CodeTemplate that overrode the Validate method as shown below:

```
public override void Validate() {  
}
```

you will need to update the method signature to the following:

```
public override System.Collections.Generic.IEnumerable<ValidationError> GetCustomValidationErrors()  
{  
}
```

After the method signature has been changed, you will need to update the code implementation to return a list of validation errors.

This change ensures that the CodeTemplates State property will always be correct and any exceptions thrown in `GetCustomValidationErrors()` will not affect the generation process.

### Updates to `IMergeStrategy.Merge()` method signature

If you are using a custom [Merge Strategy](#) please update the Merge method signature as shown below:

```
public string Merge(CodeTemplate context, string sourceContent, string templateOutput) {
    return templateOutput;
}
```

you will need to update the method signature to the following:

```
public string Merge(CodeSmith.Engine.MergeContext context)
    return context.OutputContent;
}
```

After the method signature has been changed, you will need to update the code implementation to use the various properties located on the MergeContext.

These changes were made to provide additional information for Merge Strategies (E.G., [InsertClass Merge Strategy](#)) as well as additional information to be added to the MergeContext object in the future without breaking the API.



Template Frameworks like PLINQO have already been updated for these changes and will not require a recompile!

### Encoding changes

Templates are now generated as UTF-8, this change was made to be more consistent with various other editors like Visual Studio. If you wish to have files generate using ascii or a different encoding please set the [ResponseEncoding CodeTemplate Directive](#) attribute on your master template before generating. This also ensures that your template output will always be consistent when generating in any culture.

### SchemaExplorer Type Converter changes

The SchemaExplorer Type Converters have been removed and replaced with a single [SchemaObjectFactoryTypeConverter](#) object. This new Type Converter handles the conversion for all SchemaExplorer types deriving from SchemaObjectBase or SchemaObjectCollection. If your template code is using a SchemaExplorer Type Converter as shown below it can be safely removed.

```
[TypeConverter(typeof(SchemaExplorer.TableSchemaCollectionTypeConverter))]
```

## Upgrading from CodeSmith Generator 2.x

Please read this guide when upgrading from CodeSmith Generator 2.x. CodeSmith Generator Templates are almost 100% compatible with CodeSmith Generator 2.x templates, there are a couple breaking changes that CodeSmith Generator 2.x users should be aware of when upgrading. You may need to make minor changes to your CodeSmith Generator 2.x templates to have them work perfectly in the latest version of CodeSmith Generator.

### CodeTemplate Directives are required

CodeSmith Generator requires every template to have a [CodeTemplate directive](#). The CodeTemplate directive must be the first thing in the file, with the possible exception of template comments.

### Changes to template comments

CodeSmith Generator 2.x allowed you to use several formats for template comments that the latest version of CodeSmith Generator does not allow. In particular, these two formats are no longer accepted for template comments:

```
<% // some template header %>
<%-- some comment %>
```

In the latest version of CodeSmith Generator, the only acceptable format for template headers and comments is as follows:



```
<%-- some template header --%>
<%-- some comment --%>
```

Note that other formats are still valid when you want to include a comment in the generated source code. This is distinct from including a comment in the template that does not appear in the generated code. To generate a C# comment, you still use the format

```
<% // some C# comment %>
```

and to generate a VB comment, you still use the format

```
<% ' some VB comment %>
```

### Register sub-templates declaratively

Although you can still programmatically compile sub-templates, it is more efficient to use the new [Register directive](#) instead.

## Upgrading existing Property Set Xml Settings

In previous versions of CodeSmith Generator prior to [CodeSmith Generator 4.0](#), CodeSmith Generator stored property settings in an Xml document. An entirely new format was built to replace the existing format; the new format is named [CodeSmith Generator Project file](#). It has many improvements and capabilities over the previous format. A command line option was built to upgrade existing stored Xml Property Settings to the new format.

### Upgrade Instructions

To upgrade to the new format, we will be using the [CodeSmith Generator Console application](#). The first step is to launch command prompt and type in:

```
cs.exe mypropertyset.xml /upgrade:mynewcsp.csp
```



If you are encountering any issues while upgrading to the new format, please contact [support](#).

## Introduction and Tutorials

For our Introduction and Tutorials of CodeSmith Generator we offer the below sections.

### Main Features

There are many features available in CodeSmith Generator. In our Main Features page you will see the features that really set CodeSmith Generator apart and how powerful it can be.

### What's New

Check out our What's New section to see the features that we added for each new version.

### Tutorials

Tutorials is for any CodeSmith Generator user and was designed to help get our users up and running in no time flat.

### Main Features

At its most basic, CodeSmith Generator is an application to generate code by combining templates with metadata. Within that framework, it includes a number of powerful features:

- A [powerful template language](#) similar to ASP.NET.
- A [simple user interface](#) for quick interactive code generation
- An object model that allows your templates to interact directly with the CodeSmith Generator engine
- A complete integrated development environment (IDE) for CodeSmith Generator templates
- Strong Integration within [Visual Studio](#) for executing and managing your code generation.
- Powerful code generation automation using [CodeSmith Generator Projects](#), console-based code generation, and MSBuild task.
- Interactive [debugging](#) features for tracking down template errors
- Flexible [metadata providers](#) including .NET types, database connectivity, XML support, and custom metadata sources
- [Console-based code generation](#) for use in automated build processes
- And many more features which can be found [here](#).

## What's New

For a full change log of all the new features and bug fixes for each version of CodeSmith Generator be sure to look at the [CodeSmith Generator Release Blog](#).

### CodeSmith Generator 6.0

- **Brand new template editor integrated right into Visual Studio 2010! [CodeSmith Generator Studio has been removed as the Template Editor is now integrated into Visual Studio 2010. We are also working on a standalone editor!](#)**
- Vastly improved IntelliSense with support for directives, extension methods, lambdas, generics, anonymous types, parameter information and more!
- [Improved Syntax Highlighting and Template Output Highlighting support.](#)
- Brand new parsing engine that should provide much better template errors as well as a great foundation to build on for the future.
- .NET 4.0 support in templates.
- [PropertyGrid has been updated to allow property filtering, collection editing, default instance creation, auto expanding of objects and much more.](#)
- [Added Go to Definition and View Code support!](#)
- SchemaExplorer collections have been updated to use generic collections that give a bunch of new features like LINQ support.
- New default property serializer that will enable serialization of just about any object and not require custom property serializers to be written. You can now just create an object in the template and use it as a property type.
- [Brand new Template Explorer that provides complete shell context menus and other features.](#) If you are using something like Tortoise for version control, you will now have access to those features right inside of Template Explorer.
- 64bit assembly support.
- Most of the engine is multi-threaded and should make better use of multiple core machines.
- [New Visual Studio Item templates to help you create templates faster.](#)
- Added Code Navigation support.
- [Unified Generation Experience!](#)
- [Improved documentation for Generator 6.0.](#)
- Added the ability to automatically generate Xml documentation inside of your templates by typing "" or /// before a property or method.
- Many other [small improvements](#) and bug fixes.

### CodeSmith Generator 5.3

- Added CS\_IsUserDefinedTableType as an ExtendedProperty to the SqlSchemaProviders ParameterSchema object. This will return true if the type is a User-Defined Table Type.
- Added support for Function-Based Indexes in the OracleShcemaProvider.
- Added CS\_IndexType and CS\_ColumnExpression as an ExtendedProperty to the OracleSchemaProviders IndexSchema object.
- Added the ability to save property enumerations that do not have a default value of 0 defined.
- Added the SQLAnywhereSchemaProvider. This has been tested against Sybase IAnywhere 11.0.
- Added the ISeriesSchemaProvider. Requires iSeries OS v5.4 or greater and has been tested against v6.1.
- Added Flash support to the CodeSmith Generator Studio browser.
- Added the ability to silently uninstall CodeSmith Generator using the /quiet flag.
- Added initial support for GetExtendedProperties and GetCommandResultSchemas to the PostgreSQLSchemaProvider.
- Updated the DbType and native type mappings in the PostgreSQLSchemaProvider.
- Updated the ADOXSchemaProvider error handling to better support Microsoft Visual FoxPro 9.0.
- Renamed CodeSmith to CodeSmith Generator and updated product logos, license agreements and start page.
- Updated the Microsoft Connection String Designers to the latest version.
- Updated the SqlSchemaProvider to support Table UDT's as a ParameterSchema.

### CodeSmith Generator 5.2

- Added Visual Studio 2010 Beta 2 Support (Beta).
- Added option to the installer to choose a different sample folder.
- Fixed a bug where the following error would occur: Unable to cast object of type 'SchemaExplorer.ADOXSchemaProvider' to type 'SchemaExplorer.IDbSchemaProvider'.
- Fixed a bug where multiple licenses would not be deactivated during deactivation.
- Added upgrade support for previous versions of CodeSmith.
- Fixed a bug where an exception would be thrown when an invalid connection string would be passed into any connection string editor.
- Fixed a bug where optional merged properties in a Class Library were not being marked as optional.
- Fixed a bug where selecting a blank DataSource from a UI Picker would throw a NullReferenceException.
- Added a detailed error message to FileNameEditor, it will now let the user know that there GetFileName override is throwing an

exception.

- Fixed an issue where the property grid would be blank but as soon as you clicked on the property it would appear.
- Fixed a rare bug where CodeSmith Studio couldn't resolve a template's referenced assemblies on the very first load.
- Fixed a bug where the Property Grid wouldn't refresh properties that had been changed in an assembly.
- Fixed a bug where the Splash Screen would attempt to be closed during a race condition causing an Exception.
- Fixed a bug where the Uninstaller would not close the registry key it opened.
- Fixed a bug where a Required Property would be ignored when generating from CodeTemplateGenerator.
- Added an option to the tool bar in CodeSmith Studio to add a new blank template.
- Updated CodeTemplateGenerator to use the template cache when clicking the build button.
- Fixed a bug where clicking Build in the CodeTemplateGenerator dialog would discard property data.
- Added a notification to the CodeTemplateGenerator when generating a template that's OutPutType is set to none.
- Added support for SQL Functions (table-valued and scalar-valued).
- Added IncludeFunctions Property to SchemaObjectBase, setting this to true enables SQL Function support.
- Fixed a bug in DatabaseSchemaSerializer where a changed property would never be changed back to true (DeepLoad).
- Added Filtering support to all Command UI dialogs.
- Added support to set the Command UI DataSource based on the IncludeFunctions property.
- Fixed various memory leaks greatly reducing CodeSmith's footprint.
- Fixed a bug where cached data would be deleted prematurely.
- Fixed a bug where renaming a folder shortcut to a previous name would throw a NullReferenceException.
- Added overloads for CommandResultColumnSchema and ParameterSchema to GetVBVariableType and GetCSharpVariableType in the SQLCodeTemplate and VBSqlCodeTemplate.
- Fixed a bug that prevented the Visual Studio Integration from unlocking referenced assemblies.
- Fixed a bug where the Active Snippets configuration dialog was not working as expected.
- Fixed a bug where the Active Snippets Configuration would throw an ArgumentOutOfRangeException when arranging Arguments.
- Fixed a bug where one could not activate CodeSmith inside of Visual Studio.
- Fixed a bug where generating a file to a directory outside of the Visual Studio Solution would throw an exception.
- Fixed a bug where closing Visual Studio during generation would throw a NullReferenceException.
- Fixed a bug where an XML Namespace error would occur in Visual Studio when generating .NET 3.5 console applications.
- Fixed a bug where Saving a Setup Project file during generation would throw a COMException.
- Fixed a bug where the DataSource manager would throw an exception when loading up the VistaDBSchemaProvider.
- Fixed a bug in the SQLSchemaProvider where the where logic of the command queries would return incorrect results.
- Removed the permissions (id) check in SQLSchemaProvider GetCommand's queries.
- Fixed a bug in the SQLSchemaProvider where GetCommandResultSchemas was not correctly handling temp tables.
- Fixed a bug in the SQLSchemaProvider could throw a NullReferenceException on invalid extended table data.
- Fixed a bug in the SQLSchemaProvider where connecting to a replicated database would cause a timeout to occur.
- Fixed a bug in the PostgreSchemaProvider where multicolumn indexes were not handled correctly.
- Updated the PostgreSQLSchemaProvider assembly references to the latest version.
- Fixed a bug in the SqlCompactSchemaProvider where ROWGUIDCOL was not included in the GetTableColumns().
- Fixed a performance bug in MySQLSchemaProvider, where DataReader.NextResult() or DataAdapter.Fill() would take 10-20 seconds to return data.
- Fixed a bug in the OracleSchemaProvider where the HasExtendedPropertiesTable could return a false positive.
- Updated the custom property examples: added a Collection and Drop Down example.
- Updated the Command Wrapper templates to support SQL Functions as well they can be used in Visual Studio integration.
- Added an HTML photo gallery example.
- The Kinetic Framework - Updated to latest version.
- CSLA - Updated to version 1.1.1.
- PLINQO - Updated to version 4.0.
- NHibernate - Updated to version 1.1.5.
- Updated the CodeSmith documentation.
- Various other minor changes.

## CodeSmith Generator 5.1

- Fixed a bug where removing a data source from Database Explorer wouldn't permanently remove the data source.
- Fixed a threading error when removing a data source from Database Explorer.
- Fixed a bug where CodeSmith would throw an exception when it couldn't access the systems registry.
- Fixed a bug where Copy Properties would throw an exception when called on a unsaved template.
- Fixed a bug where the SqlCompactSchemaProvider connection string builder class could corrupted additional connection string options.
- Fixed a bug where the SqlCompactSchemaProvider timestamp/rowversion columns were returning a "rowversion" native type name, should be "timestamp".
- Added CodeSmith Customer Improvement Program.
- Various other minor changes.
- Fixed a bug where Generate Outputs would throw an error if a Visual Studio Solution contained a Setup and Deployment project.
- Various minor updates to Visual Studio's Integration.
- Updated Visual Studio Integration to unlock assemblies after generation.
- Fixed a bug where a CSP in Solution folder causes ERROR: Object reference not set to an instance of an object.
- CodeSmith Studio now requires that .NET 3.5 SP1 to be installed.
- Fixed a bug where CodeSmith Studio would attempt to save a csp for a unsaved template.
- Fixed a bug where a NullReferenceException would be thrown when toggling the properties window when no template properties existed.
- Fixed a bug where extracting mapping files could cause an exception.
- Added Widening, Narrowing, Like, Let, CUInt, CULng, CUShort, and Operator to the VB.NET keyword list.
- Added var to the C# keyword list.
- Fixed a bug when using Intellisense and Math. or some variable names would throw an ArgumentOutOfRangeException.
- Updated CodeSmith Options dialog's.

- Added support to give feedback and send detailed error information from within CodeSmith.
- Updated Menu in CodeSmith Explorer, Users can now view the mapping editor, submit feedback, help, or configure options.
- Updated Manage Outputs and child dialogs to save the window dimensions.
- Added IndexedEnumerable, this is used to smartly enumerate collections and get a IsEven, IsLast, IsFirst property.
- Added Linq Querying support to all SchemaExplorer Collections.
- Added MergeProperty functionality for parsing properties from a CodeTemplate that inherits from an assembly.
- Added Insert Class Merge Strategy.
- Added CodeParser.
- Added support to detect an embedded SDK License.
- CodeSmith Configuration no longer uses xml files.
- Updated the documentation for IDbSchemaProvider and DataObjectBase.
- Fixed a bug in OracleSchemaProvider where AllowDBNull would always be set to true for view columns.
- Fixed a bug in OracleSchemaProvider where the TableSchema.PrimaryKeys collection wasn't being populated correctly.
- Updated OracleSchemaProvider's configuration to be configurable via the options dialog.
- Added SQL CLR Support to the SqlSchemaProvider. To see if a command is a CLR procedure check the "CS\_IsCLR" extended property.
- Fixed a bug in SQLSchemaProvider where an xml index would be set to null after upgrading a SQL Server 2005 database to SQL Server 2008.
- Fixed a bug in SQLSchemaProvider where the ExtendedData query was missing the PropertyBaseType and Minor columns when querying SQL Server 2000 ExtendedData.
- Added PostgreSQLSchemaProvider, SqlCompactSchemaProvider, SQLiteSchemaProvider, VistaDBSchemaProvider.
- Updated .netTiers to version 2.3 RTM.
- Updated PLINQO to version 3.0.
- Added CSLA Beta templates.
- Various other minor changes.

## CodeSmith 5.0

- Added a tab for editing variables in the CodeSmith Project settings dialog.
- Made it so that any .csp variables are automatically used when there is a string matching the variable value in the property values.
- Made it so that variables are automatically created for connection strings stored in .csp files so that the connection string isn't repeated.
- Made schema explorer designers load their data sources async so that the UI would not lock.
- Changed all SchemaExplorer designers to display in Object (Owner) format so that you can type the first couple letters to jump to the object you are looking for.
- Added ability to override plural/singular forms of words to the StringUtil.ToPlural and StringUtil.ToSingular methods.
- Added ability to specify Filter="SomeTableSchemaProperty" on ColumnSchema directives which will filter the list of columns in the designer based on the table selected in the specified property.
- Added new RegisterReference method to CodeTemplate to indicate which assemblies your generated code relies on so that they can be automatically added in Visual Studio.
- Added a menu item for managing data sources to the Visual Studio CodeSmith menu.
- Various improvements to the OracleSchemaProvider including full extended property support.
- Changed ColumnSchema designer to use a treeview so that all columns for all tables aren't loaded at once.
- Added ability to deep load all schema information at once which results in huge performance improvements. This is used by setting the DeepLoad attribute on any SchemaExplorer property in your template. This would typically be used when you know you are going to use all of the schema information from a database.
- Ability to use .net 3.5 features in templates including LINQ. This is accomplished by setting the CompilerVersion attribute on CodeTemplate to "v3.5".
- CodeSmith Projects now have a single file output mode to generate all template outputs into a single file.
- Added Ability to generate individual project outputs.
- Made it so that files being generated from a CodeSmith Project are checked out of source control before being edited.
- Improved the custom tool upgrade process so that it works 100% in all scenarios without having to make manual changes afterward.
- CodeSmith Projects can now add files to Visual Studio as code behind files to other generated files.
- CodeSmith Projects can now set a generated files build action.
- Added ability to resolve assemblies located in paths relative to the template now using Path attribute. Looks in template folder and \bin folder by default.
- Optimized template caching algorithm allows for much improved performance.
- Templates use partial classes now so you can have partial class code behinds and have access to template properties from the code behind file.
- Added GetPropertyAttribute and SetPropertyAttribute to CodeTemplate.
- Property attribute values are added for any non-recognized attributes on Property, XmlProperty and CodeTemplate directives.
- Re-organized all sample templates and projects into a more logical folder structure.
- Added new NHibernate templates in both C# and Visual Basic.
- Made various improvements to the Plinqo templates.
- .netTiers updated to the 2.3 version of the templates.
- Added VB versions of many sample templates and projects.
- Included the latest version of the NuSoft framework templates.
- Many other minor enhancements, performance improvements, and bug fixes.

## CodeSmith 4.1

- Auto property refresh when running your templates including SchemaExplorer objects and external XML sources.
- Added IDbConnectionStringEditor interface so schema providers can provide connection string editing interfaces. A connection string editor was implemented for SqlSchemaProvider, ADOXSchemaProvider and OracleSchemaProvider.
- Added Indexes and Keys to the SchemaExplorer tool window in CodeSmith Studio.
- Added support for Visual Studio 2008 (Orcas).

- Added NoWarn attribute to CodeTemplate directive to allow ignoring compiler warnings.
- Added several new sample templates as well as source code for the SqlSchemaProvider.
- Many other minor enhancements, performance improvements, and bug fixes.

## CodeSmith 4.0

- [CodeSmith Projects \(.csp\)](#) - This feature makes automating your code generation process really easy and consistent whether you are working from inside of Visual Studio 2005, MSBuild, Windows Explorer, a command line / batch file, or CodeSmith itself.
- [ActiveSnippets](#) - Imagine Visual Studio 2005 snippets, but with the full power of CodeSmith available to execute any logic or access any complex metadata (including database schema and XML data) to control the output of your snippets.
- [CodeSmith Maps \(.csmmap\)](#) - This feature will allow you to create dictionary style maps of things like SQL to C# data type mappings.
- [.netTiers 2.0](#) - The .netTiers templates have been greatly enhanced and included with CodeSmith 4.0.
- [NHibernate Templates](#) - NHibernate templates have now been included and are able to get you started with using NHibernate.
- [CSLA .NET 2.0 Templates](#) - They latest CSLA .NET 2.0 templates have been included and are greatly improved.
- [DbSnapshot Templates](#) - Script all objects and table data out from a Microsoft SQL Server database.
- [Extended Property Management](#) - You can now manage schema extended properties inside of CodeSmith Studio.
- [Property Persistence](#) - CodeSmith now remembers the property values from the last time you executed a template.
- Greatly enhanced Visual Studio 2005 integration.
- Support for running CodeSmith in non-admin accounts as well as in Vista UAC.
- CodeSmith Studio
  - Many improvements have been made to the performance of the IDE.
  - IntelliSense has been improved (including ctrl-space support) in both templates and code behind files.
  - Added recent news items to the start page.
- CodeSmith Explorer
  - Supports drag and drop to move/copy files.
  - Improved performance.
- CodeSmith Engine
  - PropertyChanged event is now exposed on each template.
  - Added OnChanged attributes to Property and XmlProperty directives.
  - XmlProperty now stores a file reference to the source XML instead of the XML contents.
  - XmlProperty now shows the XML file name in the property grid and can be edited.
  - Added ContextData object to templates for storing various non-persistent state which is shared with sub-templates.
  - IPropertySerializer interface has been changed to give property serializers access to more contextual information.
  - Added a Initializing state to the template State enum.
- SchemaExplorer
  - Improved handling of SQL Server BLOB DataTypes
  - Improved ParameterSchema meta-data in CommandParameterSchema now containing DefaultValue Schema Information.
  - Ton of new system extended properties added to SchemaExplorer objects.
- Many other minor enhancements, performance improvements, and bug fixes.

## CodeSmith 3.2

- Built and optimized for .NET 2.0 / Visual Studio 2005
- CodeSmith MSBuild task
- Numerous other minor improvements and bug fixes

## CodeSmith 3.1

- New help file
- Numerous other minor improvements and bug fixes

## CodeSmith 3.0

- Completely re-written parser/compiler which is faster and correctly reports line numbers from the template instead of from the compiled template source. This results in a much nicer debugging experience.
- [XML support](#) - There is now an XmlProperty directive that makes working with XML *much* easier. This directive will give you a strongly typed object model to work with if you provide an XSD schema or it will give you an XmlDocument instance if you don't. This feature combined with the new IntelliSense feature make working with XML a breeze.
- [Statement completion](#) in CodeSmith Studio (similar to Visual Studio's IntelliSense)
- [Template caching](#).
- New [Register directive](#) that makes working with sub-templates much nicer.
- Console client has been improved to include a batch mode, [setting properties from the command line](#), and the ability to use any [merge strategy](#).
- [Merge strategies](#) have been overhauled to be more extensible and can be setup to work with any language.
- New [PreserveRegions merge strategy](#) has been introduced.
- DbDocumenter templates have been overhauled to be a best-practices sample for 3.0.
- [Indented output rendering](#).
- IPropertySerializer interface can be implemented to allow for [serialization of custom property types](#).

- New [PreRender](#) and [PostRender](#) methods that can be overridden in your templates.
- Ability to [auto-execute SQL scripts](#) after generating them.
- Ability to [render to more than one TextWriter](#) at a time.
- Tons of other minor improvements and bug fixes.

## Tutorials

The CodeSmith Generator Tutorial includes the various sections which will quickly help you become a master of Code Generation. Although there's a lot of depth to CodeSmith Generator, you can get started with it quickly. In this section of the documentation, we'll walk you through a few common code generation scenarios.

### Getting Started

The best way to understand the power of CodeSmith Generator is to try it out. Although CodeSmith Generator has many advanced features, you can begin using it to help produce code without mastering all of those features. In this section, you'll learn how to use Generator to generate a useful piece of utility code - specifically, a strongly-typed hash table class.

### Writing your first Template

Knowing how to execute templates that others have written is the first step towards [getting started](#) with CodeSmith Generator, but to realize the full benefit of CodeSmith Generator in your day to day development tasks, you'll need to write your own templates. In this tutorial, you'll learn how to do just that, working through the entire process of writing a CodeSmith Generator template from start to finish.

### Write a Template with Database Metadata

One of the key uses for code generation is to build code based on database schema. CodeSmith Generator enables this scenario through the use of the SchemaExplorer assembly, which provides types for working directly with SQL Server or ADO data as well as designers that can be used to access those types from CodeSmith Generator. In this tutorial, you'll see how you can use the information available through SchemaExplorer, together with scripting code, to make short work of building a complex T-SQL script.

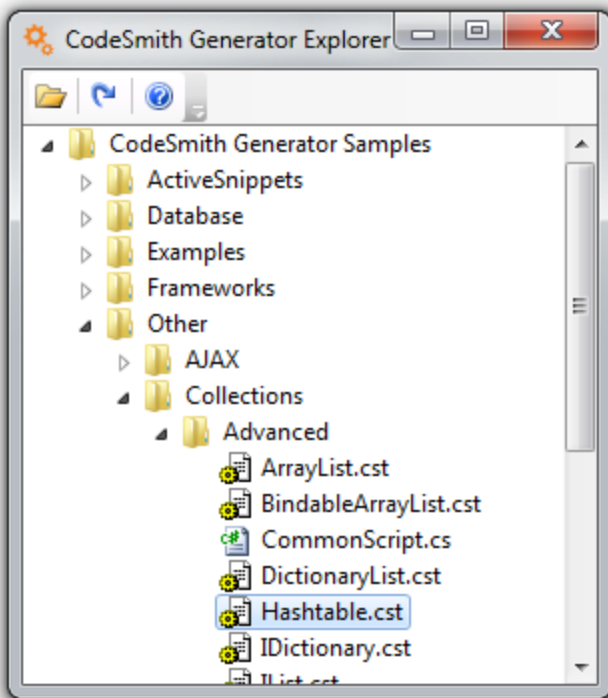
## Getting Started

The best way to understand the power of CodeSmith Generator is to try it out. Although CodeSmith Generator has many advanced features, you can begin using it to help produce code without mastering all of those features. In this section, you'll learn how to use Generator to generate a useful piece of utility code - specifically, a strongly-typed hash table class. This exercise should take you no more than five minutes to complete, but it will introduce you to [Template Explorer](#), and show you the power of Generator's template-based code generation scheme.

Next: [Launching Template Explorer](#)

### Launching Template Explorer


One way to start a code generation session is with Template Explorer. Just as Windows Explorer serves to organize files and folders stored on your computer, Template Explorer serves to organize templates. To launch Template Explorer, select **CodeSmith Generator Explorer** from the CodeSmith Generator program menu. This will open Template Explorer with an initial view showing all of the folders containing templates in your CodeSmith Generator installation.



**Important Note:** Generator Explorer is the name of the application as seen in the tool bar. However, it is just a wrapper around Template Explorer.

Check out this video tutorial for an overview on the Template Explorer:

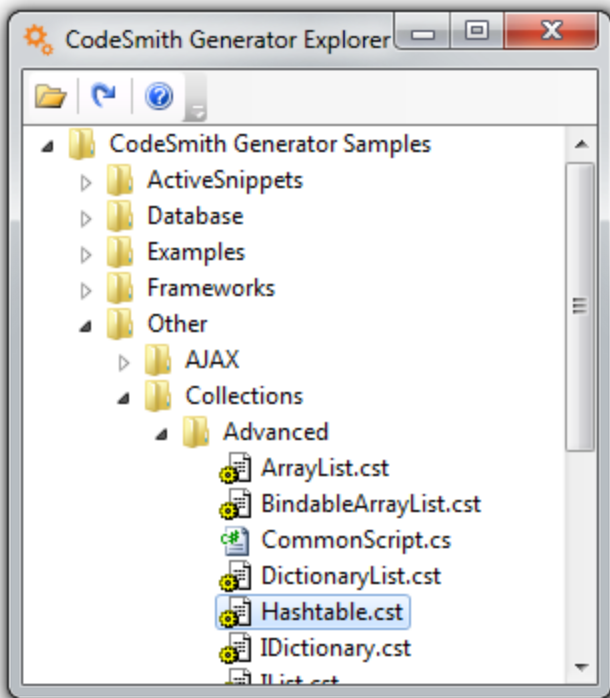
Next: [Opening a Template](#)


 If you have not yet activated your copy of CodeSmith Generator, you'll see a product activation dialog box when you launch CodeSmith Explorer. Click [Try](#) to proceed with this example, or [Register](#) to proceed with [product activation](#).

## Opening a Template

Templates are patterns for generated code. CodeSmith Generator comes with a set of useful templates to get you started. You can also download more templates from our [CodeSmith Community Site](#) as well as share them.

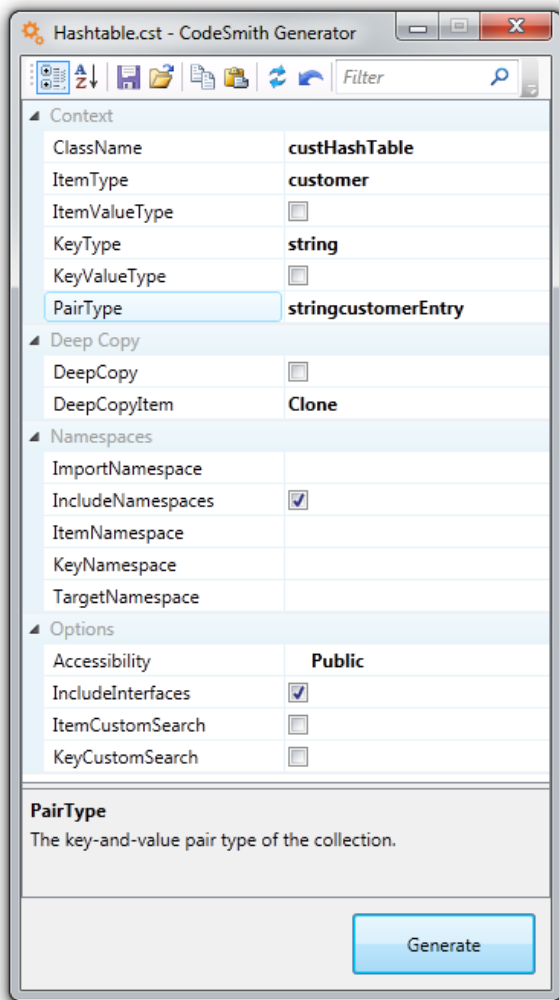
You might find that as you work with CodeSmith Generator, you'll start to develop your own custom templates. Template Explorer makes it easy to generator a template quickly as well as organize templates in folders so you can quickly find the template you are looking for.



 To open a folder, click on the arrow sign to the left of a specific folder to see the list of templates stored in the folder.

The .cst file extension stands for "CodeSmith Generator Template." You can probably guess from the names what the various templates do. For example, the HashTable.cst template, which can be found in the Other\Collections\Advanced folder, generates C# code for a hash table class. To open this template, just Double-click the template or right-click the template and select the Execute menu item.

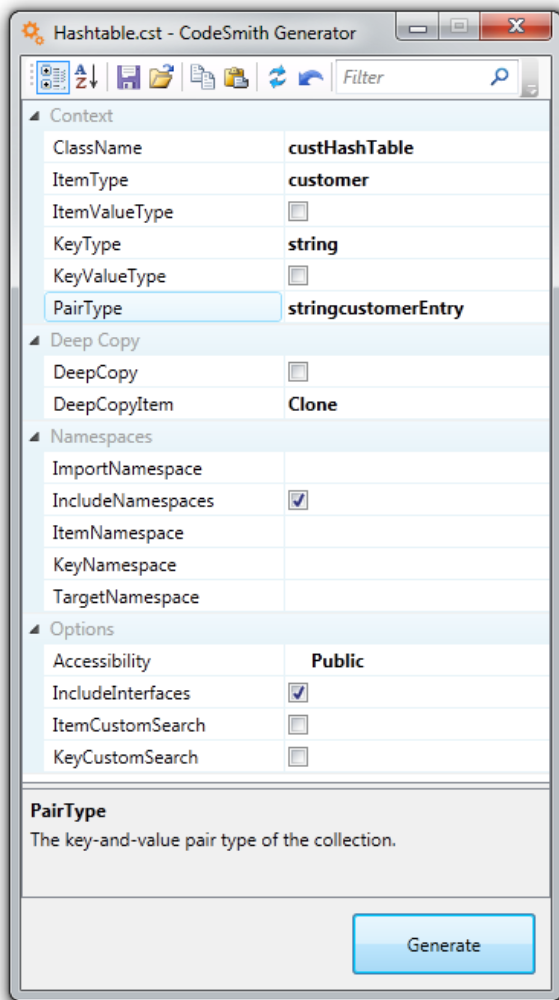




Next: [Setting Properties](#)

## Setting Properties

A code generator that generated the exact same code every time wouldn't be very useful (you might as well just paste in a saved code file, if that's what you want). CodeSmith Generator templates use *properties* to let you customize the generated code. When you open a template from Template Explorer, the template's property sheet shows you all of the properties that the template requires. You need to supply values for these properties before CodeSmith Generator can generate the code for you. The Hashtable.cst template that we opened in the previous step requires four string properties (ClassName, ClassNamespace, ItemType, and KeyType) and one enumerated property (Accessibility). You can type any value you like for a string property; an enumerated property presents you with a drop-down list of choices when you click in it. For this first experiment, fill out the property sheet this way:



One of the best things about CodeSmith Generator is that properties can be based on many different types of metadata. For instance, you can create a property that presents a list of all of the tables in a database, letting the user choose a table when they're generating the code. You can learn more about this in the section on [Driving Templates with Metadata](#).

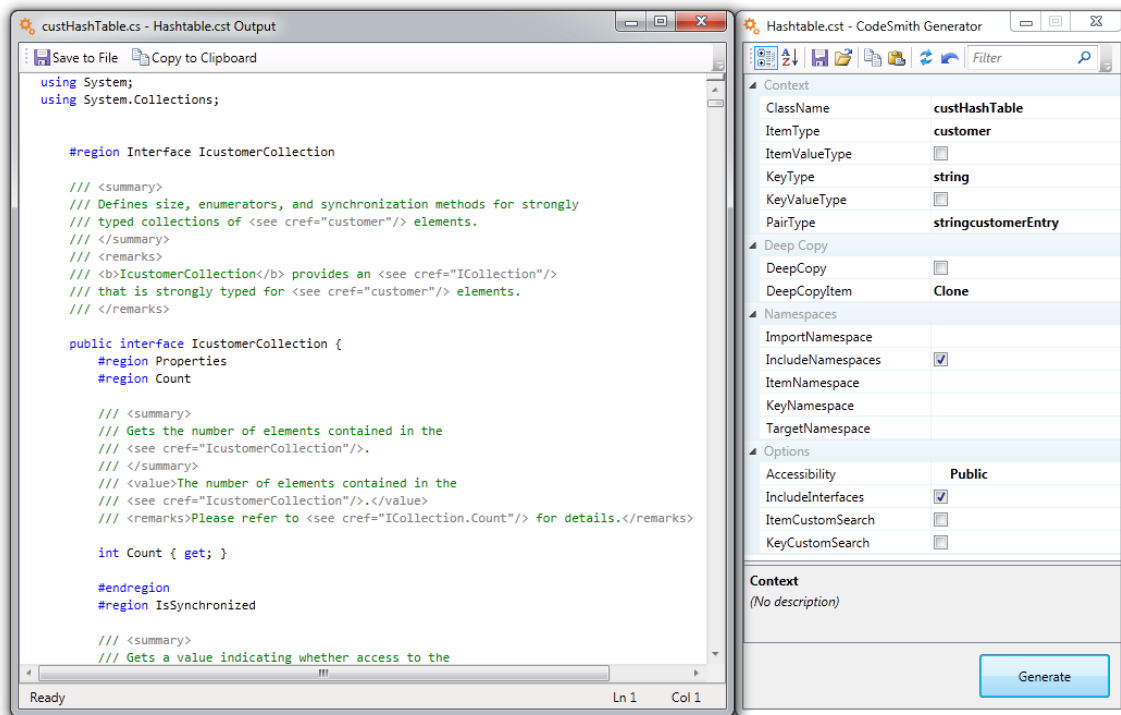
**i** You can also filter the properties that are shown in the property sheet, by typing the name of the property in the Filter Search Box located above the property sheet.

Next: [Generating Code](#)

## Generating Code

When you've finished setting properties for the template, you're ready to generate code. To do this, click the **Generate** button at the bottom of the template's property sheet. CodeSmith Generator will take the property values that you entered and combine them with the template to create the code, and display it in an [output window](#). The generated code can be edited by typing in the generated output window. You can also easily copy or save the generated text by clicking on the respected buttons at the top of the output window.

In this case, the code window contains around 320 lines of generated code for the hash table class, implementing the `IDictionary`, `ICollection`, `IEnumerable`, and `ICloneable` interfaces. There's nothing there that you couldn't write yourself if you're a reasonably experienced C# developer - but why bother? This is the sort of routine work that CodeSmith Generator is ideally suited for. With CodeSmith Generator, you can devote your time and energy to identifying patterns in your code, turning them into templates, and then reuse them with maximum flexibility in the future.

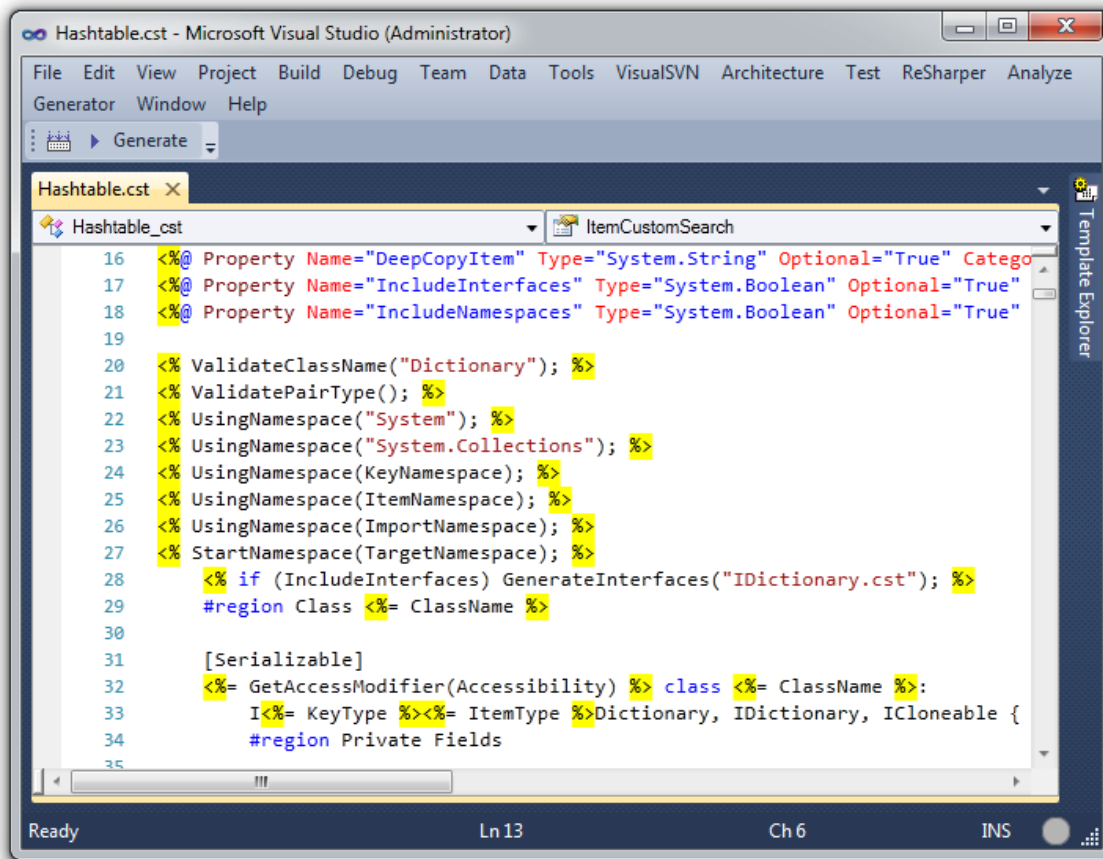


Next: Inspecting the Template

Advanced: Using a CodeSmith Generator Project to Execute CodeSmith Templates from Anywhere.

## Inspecting the Template

Let's take a peek behind the scenes at the HashTable.cs template itself. Remember, the template contains the instructions that CodeSmith Generator uses to generate the code. Return to Template Explorer, but this time right-click on the template and select Edit. This will open the template in the default template editor. CodeSmith Generator ships with a full-featured Template Editor as shown below for editing or generating templates.



Later on in this help file you can learn more about the [Template Editor](#) in detail. For now, just poke around the source code for the template a bit: it's displayed in the main editing area. As you can see, CodeSmith Generator's template language is very similar to ASP.NET. The file starts off with a set of directives, including some that declare the various properties that appear in the template. These properties can be used later in the template by enclosing them in special tokens. For example, line 32 of code in the template

```
<%= GetAccessModifier(Accessibility) %> class <%= ClassName %>:
```

instructs CodeSmith Generator to output the the class modifier (E.G., public, private...) followed by literal string "class" followed by the value of the ClassName property when it is generating code.

If there's something you don't like about the HashTable template, you can change it here. For example, you might like to add some comments to explain the reason why each interface is included. You can just type these into the template and save your changes to have CodeSmith Generator use the altered template in the future.

Next: [Where to Go from Here](#)

## Where to Go From Here

That's all you need to know to generate code using the templates that are included with CodeSmith Generator:

1. Launch Template Explorer
2. Select a template
3. Supply values for properties
4. Generate the code

But you can do much more than just use the included templates. Here are some places to continue your exploration:

- Visit the [CodeSmith support site](#) for more templates
- Learn more about [template syntax](#) to write your own templates
- Get the details on editing templates with the [Template Editor](#)
- See how to [enhance your templates with metadata](#)
- Incorporate CodeSmith into your build process

## Writing Your First Template

Knowing how to execute templates that others have written is the first step towards [getting started](#) with CodeSmith Generator. But to realize the full benefit of CodeSmith Generator in your day to day development tasks, you'll need to write your own templates. In this tutorial, you'll learn how to do just that, working through the entire process of writing a CodeSmith Generator template from start to finish.

Next: [Spotting the Need](#)

## Spotting the Need

Think about your average day of software development. Some of it probably involves brand-new innovative work that breaks new ground and doesn't resemble anything that you've ever done before. But other parts are probably more routine. Whether it's writing the code for a public property backed by a private variable, creating an About This Application dialog box for a new product, or designing a new page for the corporate Web site, much of your day probably involves routine coding tasks that you've done before with only minor variations.

Any time you find yourself doing one of these repetitive tasks, you've found a candidate for code generation. Creating source code files (or Web pages, SQL statements, HTML pages, or any other text file) with minor variations is exactly the sort of thing that CodeSmith Generator is designed for. For example, if you're writing C# code, you know that every C# project contains an AssemblyInfo.cs file with metadata about the project. Visual Studio automatically creates a skeleton AssemblyInfo.cs file for you when you create a new project, but it's full of comments designed for the novice developer, and attributes for every conceivable purpose. That's fine as a teaching tool, but it's not what most developers want to see in their source code. So typically, you'll start a new project by cutting out the junk, adding a few comments of your own, and making a standard set of changes to the attributes that remain. That's a perfect candidate for code generation: a process that you do over and over again with a few variations. Let's use CodeSmith Generator to generate just the AssemblyInfo.cs file that you need, without all the fluff.

Next: [Creating the Template](#)

## Creating the Template

CodeSmith Generator templates are plain text files that contain three different types of content:

- Directives to the CodeSmith Generator engine
- Static content that is copied directly to the template's output
- Dynamic content (programming code) that is executed by the CodeSmith Generator engine

The dynamic content in a CodeSmith Generator template can be written in C#, Visual Basic, or JScript. For this template, we'll use C# as the template scripting language. We can set the scripting language (C#, Visual Basic, JScript) in the [CodeTemplate](#) directive's language attribute as shown in the code sample below). For this template you can use the [Generator Template Editor](#) or notepad.

Every CodeSmith Generator template starts with a [CodeTemplate](#) directive. This directive tells CodeSmith Generator some basic facts about the template. Here's the [CodeTemplate](#) directive for this template:

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
```

The CodeTemplate directive sample above defines three attributes (Language, TargetLanguage and Description).

- The **Language** attribute specifies the scripting language that will be used within the template itself. Their are three valid attribute values that can be defined: C#, VB or JavaScript.
- The **TargetLanguage** attribute specifies the language of the generated output.
- The **Description** attribute gives the purpose of the template.

With this single line of code saved as a file named AssemblyInfo.cst, you've got a CodeSmith Generator template. But it doesn't do anything yet.

Next: [Start with the Result](#)

## Start with the Result

The easiest way to build a CodeSmith Generator template is to start with an example of the code that you want to generate - in this case, a finished AssemblyInfo.cs file. Here's one that we'll use as we move through this tutorial:

```

using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: 1/1/1973
// Author: Blake Niemyjski
//
[assembly: AssemblyTitle("User storage utility")]
[assembly: AssemblyDescription("Helps manage data in Isolated Storage files.")]
[assembly: AssemblyConfiguration("Retail")]
[assembly: AssemblyCompany("MegaUtilities, Inc.")]
[assembly: AssemblyProduct("StorageScan")]
[assembly: AssemblyCopyright("Copyright (c) MegaUtilities, Inc.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]

```

When you're looking at the file that you want to generate, you need to break the file up into three different types of content:

- Content that will never change
- Content that can be automatically generated.
- Content that you will prompt the user for

In the sample above, we've decided that we are going to automatically generate the Created Date on line 4, and we'll prompt the user to specify the values for Author, Title, Description, Configuration, Company, Product, Version and FileVersion. The rest of the file we'll treat as static text. Of course, you need to make these decisions with an understanding of how you'll use your template. In this case, for example, we've decided to hard-code the AssemblyDelaySign attribute to always be true. If your use of that attribute varied from project to project, you would want to make that a dynamic part of the template that you prompted the user for.

Now that we know what we want to build, it's time to get the content into the template.



Although we're not using the capability in this example, CodeSmith Generator templates can easily contain conditional logic. For example, you could prompt the user for a value for the AssemblyDelaySign attribute, and then include additional attributes in the template's output if they set that attribute to true.

Next: [Static Content in the Template](#)

## Static Content in the Template

Adding static content to a CodeSmith Generator template is easy. If CodeSmith Generator sees something in the template that it doesn't recognize as dynamic scripting content, it copies that content directly to the template's output. So the first step in building our new template is to tack the existing file on to the template without any changes:

```

<#@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: 1/1/1973
// Author: Blake Niemyjski
//
[assembly: AssemblyTitle("User storage utility")]
[assembly: AssemblyDescription("Helps manage data in Isolated Storage files.")]
[assembly: AssemblyConfiguration("Retail")]
[assembly: AssemblyCompany("MegaUtilities, Inc.")]
[assembly: AssemblyProduct("StorageScan")]
[assembly: AssemblyCopyright("Copyright (c) MegaUtilities, Inc.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]

```

At this point, you can run the template, and you'll get output: in fact, you'll get the original file back, because there's no dynamic content in this template at all. Next, you need to modify the template to take advantage of the power of CodeSmith Generator's dynamic scripting and interactive metadata.

[Next: Making the Content Dynamic](#)

## Making the Content Dynamic

The next step is to let CodeSmith Generator generate the parts of the output that it can calculate automatically. To do this, we'll insert C# code into our template, using special scripting tags with the same syntax as ASP.NET. CodeSmith Generator looks for sections of your template surrounded with `<%=` and `%>` tokens, and treats the contents of those tags as expressions to evaluate at runtime. The result of those expressions is then inserted into the generated code in place of the scripting expression.

Here's the template with two expressions in place of the hard-coded dates in the original (the changes are on line 5):

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: <%= DateTime.Now.ToLongDateString() %>
// Author: Blake Niemyjski
//
[assembly: AssemblyTitle("User storage utility")]
[assembly: AssemblyDescription("Helps manage data in Isolated Storage files.")]
[assembly: AssemblyConfiguration("Retail")]
[assembly: AssemblyCompany("MegaUtilities, Inc.")]
[assembly: AssemblyProduct("StorageScan")]
[assembly: AssemblyCopyright("Copyright (c) <%= DateTime.Now.Year.ToString() %> MegaUtilities,
Inc.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]
```

Now, the creation date and copyright date will be filled in automatically by CodeSmith Generator whenever the template is executed. But there are other parts of this file that can't be determined automatically by CodeSmith Generator, such as the assembly title and assembly description. For that sort of variable data, the solution is to prompt the user at runtime, using CodeSmith Generator properties.

[Next: Adding a Template Property](#)

## Adding a Template Property

CodeSmith Generator uses *property directives* to define the metadata for a template. You need to add one property directive to the template for each piece of information that you want to collect from the user when the code is generated. Here are the property directives we'll need for our AssemblyInfo.cst template:

```
<%@ Property Name="Author" Type="System.String" Description="Lead author of the project." %>
<%@ Property Name="Title" Type="System.String" Description="Title of the project." %>
<%@ Property Name="Description" Type="System.String" Description="Description of the project." %>
<%@ Property Name="Configuration" Type="System.String" Default="Debug" Description="Project
configuration." %>
<%@ Property Name="Company" Type="System.String" Default="MegaUtilities, Inc." %>
<%@ Property Name="Product" Type="System.String" Description="Product Name." %>
<%@ Property Name="Version" Type="System.String" Default="1.0.*" Description=".NET assembly
version." %>
<%@ Property Name="FileVersion" Type="System.String" Default="1.0" Description="Win32 file
version." %>
```

Each of these properties has a name (which we'll use to refer to the property in scripting code), and a type (in this case, they're all strings). Some of the properties also have default values, although, as you can see, you're not required to supply a default value for a property. Most of the properties also have a description. When the user selects a property in the template's property sheet, CodeSmith Generator displays the description to help them enter the proper data.

The next step is to make the connection between these properties and the spots in the template where we want to output their values.

[Next: Using Properties in the Template](#)

## Using Properties in the Template

To insert the value of a property in the generated output from the template, use the same `<%=` and `%>` syntax that you used with calculated fields, but this time use the name of the property for CodeSmith Generator to evaluate. Here's our final template:

```
<%% CodeTemplate Language="C#" TargetLanguage="C#" Description="Create an AssemblyInfo.cs file." %>
<%% Property Name="Author" Type="System.String" Description="Lead author of the project." %>
<%% Property Name="Title" Type="System.String" Description="Title of the project." %>
<%% Property Name="Description" Type="System.String" Description="Description of the project." %>
<%% Property Name="Configuration" Type="System.String" Default="Debug" Description="Project
configuration." %>
<%% Property Name="Company" Type="System.String" Default="MegaUtilities, Inc." %>
<%% Property Name="Product" Type="System.String" Description="Product Name." %>
<%% Property Name="Version" Type="System.String" Default="1.0.*" Description=".NET assembly
version." %>
<%% Property Name="FileVersion" Type="System.String" Default="1.0" Description="Win32 file
version." %>
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: <%= DateTime.Now.ToLongDateString() %>
// Author: <%= Author %>
//
[assembly: AssemblyTitle("<%= Title %>")]
[assembly: AssemblyDescription("<%= Description %>")]
[assembly: AssemblyConfiguration("<%= Configuration %>")]
[assembly: AssemblyCompany("<%= Company %>")]
[assembly: AssemblyProduct("<%= Product %>")]
[assembly: AssemblyCopyright("Copyright (c) <%= DateTime.Now.Year.ToString() %> <%= Company %>")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("<%= Version %>")]
[assembly: AssemblyFileVersion("<%= FileVersion %>")]
[assembly: AssemblyDelaySign(true)]
```

Note that a single property (such as Company) can appear at multiple places in the template.



You can download the above template by clicking [here](#)

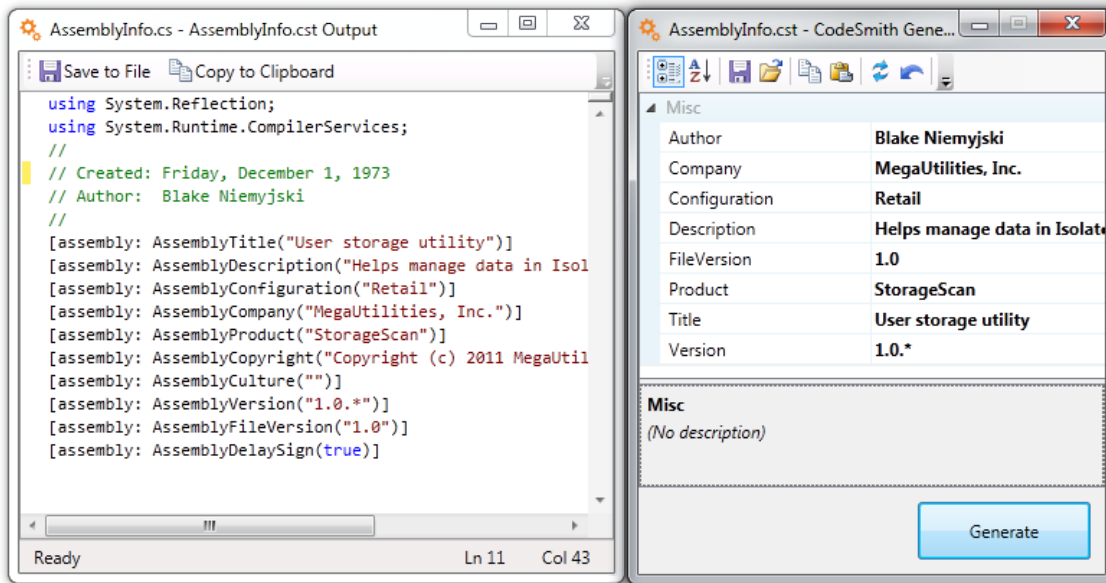
By now the template might look a good deal more complicated to you than the original file. But remember: you only have to write the template once. Then you just use it whenever you need a new file. The investment in time of adding property directives and other dynamic content will be repaid very quickly as you use the template.

Next: [Compiling the Template and Generating Code](#)

## Compiling the Template and Generating Code

At this point, the template is ready to use. Save the file, and then double-click it in Windows Explorer. This will open the template's property sheet. Fill in values for the template's properties and click the 'Generate' button to build a new AssemblyInfo.cs file instantly:





There! Wasn't that easier than editing yet another file that Visual Studio didn't build to your standards?

We chose this example to be a simple "Hello World" type template. How much time it would save you depends on how many new projects you create, of course. But this same technique - starting with the output you want to build, identifying the static content, making the content dynamic, and adding properties for the dynamic content - works with a wide range of code. You can build code for remoting and Web services, data access layers, standard user interfaces, and anything else you can imagine. CodeSmith Generator lets you replace repetitive hand-coding with code generation. That's a powerful productivity booster that you'll wonder how you ever lived without.

## Write a Template with Database Metadata

One of the key uses for code generation is to build code based on database schema. CodeSmith Generator enables this scenario through the use of the SchemaExplorer assembly, which provides types for working directly with SQL Server or ADO data as well as designers that can be used to access those types from CodeSmith Generator. In this tutorial, you'll see how you can use the information available through SchemaExplorer, together with scripting code, to make short work of building a complex T-SQL script.

Next: [HTTP Endpoints in SQL Server](#)

More information:

[Using SchemaExplorer](#)

## HTTP Endpoints in SQL Server 2005

Among the many new features in SQL Server 2005 is the ability to create *HTTP endpoints* by running T-SQL code. HTTP endpoints have several uses, including setting up SQL Service Broker connections and database mirroring over TCP/IP, but the one we'll be concerned with here is that HTTP endpoints make it easy to build Web services that return SQL Server data. In fact, if your copy of SQL Server 2005 is running on Windows Server 2003, you don't even need to have IIS installed to create a Web service that returns SQL Server data. A stored procedure coupled with a CREATE ENDPOINT statement will do the trick.

As with many of the other advanced parts of T-SQL, though, the CREATE ENDPOINT statement has a good many optional clauses and a lot of complexity. If you're going to need it more than once or twice, that offers an ideal opening for code generation. Rather than deal with that complexity all the time, figure it out once and embed your knowledge in a CodeSmith template. To begin with, you'll need an example of the SQL that you want to create.

Next: [The Desired SQL Statements](#)

## The Desired SQL Statements

We're going to create an HTTP endpoint that returns all of the data from a particular table. Because HTTP endpoints can only return information from stored procedures or functions, this means we'll actually have to build two SQL statements: one to create a stored procedure, and one to create the endpoint itself. As usual, the easiest way to build a CodeSmith template is to start with a copy of the output that you want to produce. In this case, here are the SQL statements to build an HTTP endpoint based on the Person.AddressType table in the AdventureWorks sample database:

```
CREATE PROC dbo.PersonAddressTypeProc
AS
SELECT
```

```

        AddressTypeID,
        Name,
        rowguid,
        ModifiedDate
    FROM
        Person.AddressType
GO
>CREATE ENDPOINT GetAddressType
    STATE = STARTED
AS HTTP
(
    PATH = '/AddressType',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = 'localhost'
)
FOR SOAP
(
    WEBMETHOD 'AddressTypeList'
        (NAME='AdventureWorks.dbo.PersonAddressTypeProc'),
    BATCHES = DISABLED,
    WSDL = DEFAULT,
    DATABASE = 'AdventureWorks',
    NAMESPACE = 'http://AdventureWorks/AddressType'
)
GO

```

Now that we understand where we're headed, we can start the journey. This time, we'll use CodeSmith Generator as our tool, to get a sense of the support that it offers for quickly writing templates. We've highlighted two types of information in the statements above. The red highlights show parts of the SQL statements that the user can choose from a small list of possibilities. The green highlights show information that CodeSmith Generator can determine from the SQL Server database after the user specifies a database table. The rest of the template will just be static text.

Next: [Creating the Template in CodeSmith Generator](#)


Looking at the SQL

You don't really need to understand the ins and outs of the CREATE ENDPOINT statement to follow along with this tutorial, but you might like to know what's going on here anyhow. Here are a few notes on the various clauses in this SQL statement:

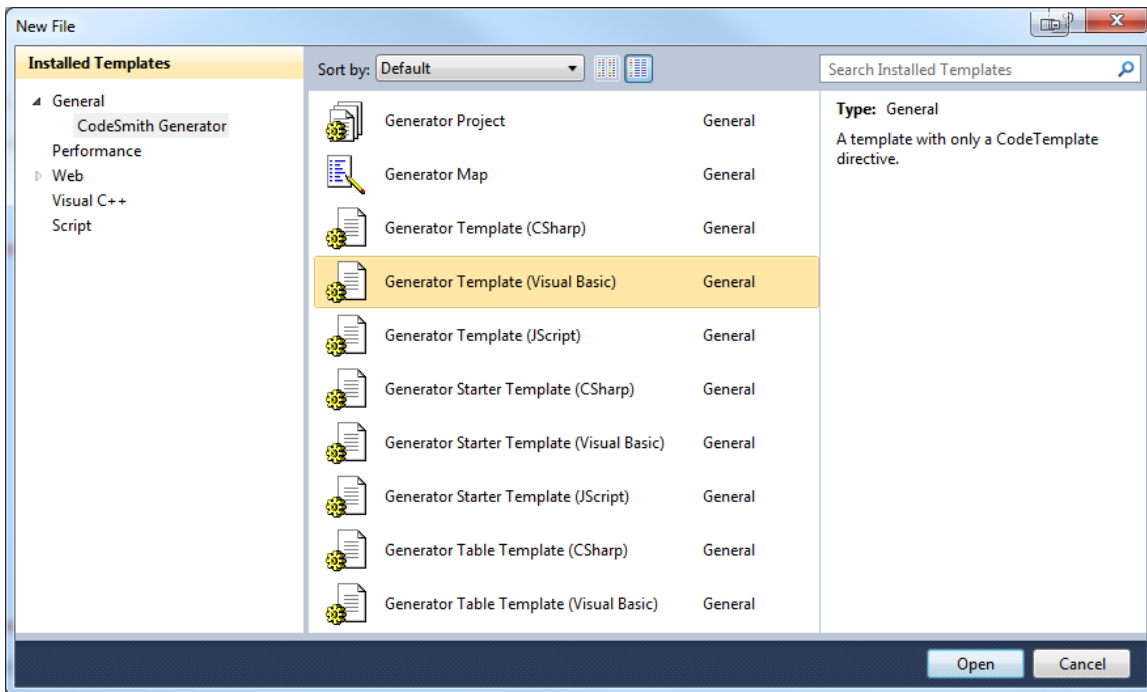
- The STATE clause specifies the initial state of the endpoint. It can be started, stopped (listening but returning errors to clients) or disabled (not even listening for requests)
- The AS HTTP clause specifies the transport protocol to use. You can also specify AS TCP here.
- The PATH clause specifies the URL on the server that clients will use to reach this Web service.
- The AUTHENTICATION clause specifies how clients will authenticate themselves to the SQL Server: BASIC, DIGEST, NTLM, KERBEROS, or INTEGRATED.
- The PORTS clause specifies whether the service will listen on the CLEAR or SSL ports, or both (other clauses, not shown here, let you specify non-standard port numbers)
- The SITE clause lets you specify a hostname for the computer that will respond to requests.
- The FOR SOAP clause states that this endpoint will respond to SOAP messages.
- The WEBMETHOD clause defines a Web method, mapping a method name to the name of a stored procedure
- The BATCHES clause specifies that this endpoint won't process arbitrary SQL statements.
- The WSDL clause specifies that it will provide WSDL support.
- The DATABASE clause specifies the database that contains the data.
- The NAMESPACE clause specifies the XML namespace for the messages.

## Creating the Template in the Generator Template Editor

This time, we'll use the [Generator Template Editor](#) to create the template. This will help get you familiar with the Generator Template Editor as well as see how fast it can speed up template development. To get started, launch Visual Studio and select File > New > File from the Visual Studio menu bar.

 You can also open a new or existing Visual Studio Project and select Add new item from the Solution Explorer.

This will open the Visual Studio New File Wizard. Next, you will want to select CodeSmith Generator under the General Installed Templates node. Doing this will only show you the available CodeSmith Generator Item Templates.



We are wanting to create a new Visual Basic Generator Template so we will choose the item above by double clicking on the selected item or clicking the Open button.

This will create a new template with some boiler plate code in it to remind you how the various parts of a CodeSmith Generator template fit together. Start by modifying the [CodeTemplate directive](#):

```
<%@ CodeTemplate Language="VB" TargetLanguage="T-SQL" Description="Create an HTTP Endpoint." %>
```

The TargetLanguage attribute is used to determine how to syntax highlight the static content of a template. The Description attribute is used to provide a tooltip for the template.

Next, replace the rest of the sample template code with the T-SQL that we want to generate. Now we've got the starting point: a template that turns out completely static SQL.

```

<%@ CodeTemplate Language="VB" TargetLanguage="T-SQL" Description="Create an HTTP Endpoint." %>
CREATE PROC dbo.PersonAddressTypeProc
AS
    SELECT
        AddressTypeID,
        Name,
        rowguid,
        ModifiedDate
    FROM
        Person.AddressType
GO
CREATE ENDPOINT GetAddressType
    STATE = STARTED
AS HTTP
(
    PATH = '/AddressType',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = 'localhost'
)
FOR SOAP
(
    WEBMETHOD 'AddressTypeList'
        (NAME='AdventureWorks.dbo.PersonAddressTypeProc'),
    BATCHES = DISABLED,
    WSDL = DEFAULT,
    DATABASE = 'AdventureWorks',
    NAMESPACE = 'http://AdventureWorks/AddressType'
)
GO

```

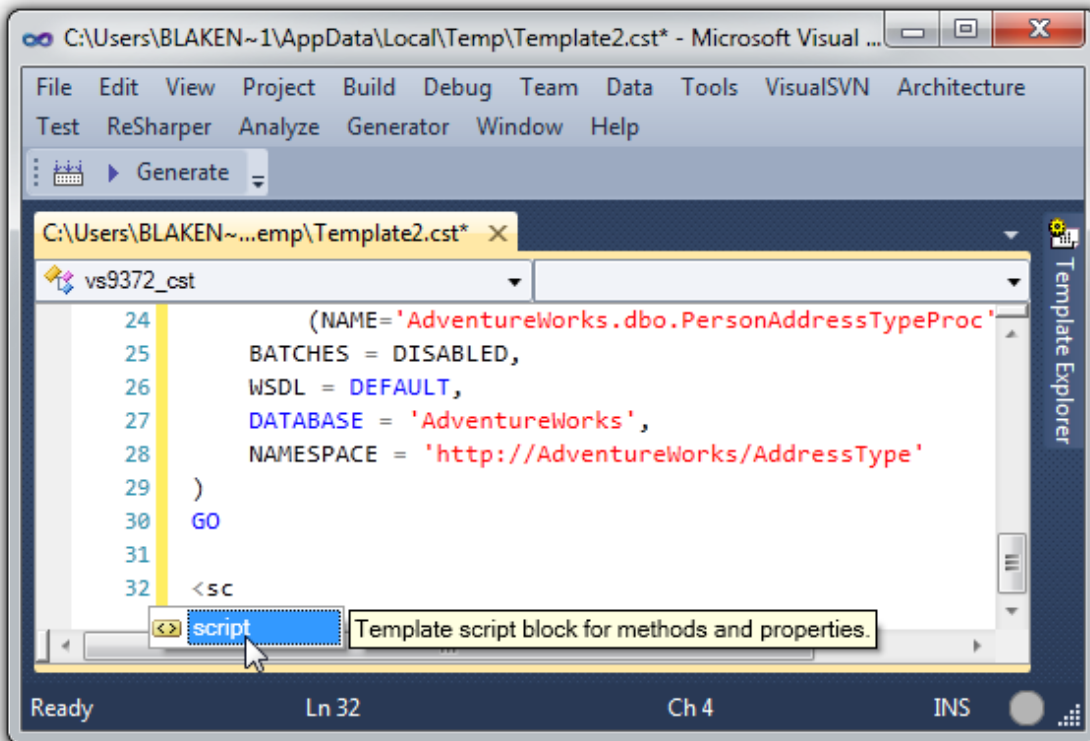
Of course, you don't want a static template. The next step is to start making the content dynamic.

Next: [Setting up Enumerated Properties](#)

### Setting up Enumerated Properties

Several of the pieces of information that we want to collect from the user have a limited number of acceptable choices. For example, the state of the endpoint can only be STARTED, STOPPED, or DISABLED; anything else will lead to a T-SQL error. Rather than prompting the user for freeform input (and running the risk of having them type an unacceptable value), it's much more sensible to offer a list of just the acceptable choices. Fortunately, you can do this by defining an enumerated property.

To set up an enumerated property, you need to define a type that only allows the values you want. You can do this by creating an enumeration. Start by moving to the end of the template and start typing <script which will show you an IntelliPrompt and allow you to autocomplete the script block.



**i** If you press the tab button or click on the script IntelliPrompt (or any IntelliPrompt) which is shown in the above image, the action will be autocompleted. In this case the script block will be created with an ending script tag.

After the script block has been created, lets create the new enumeration type:

```

<script runat="template">
Public Enum StateEnum
    STARTED
    STOPPED
    DISABLED
End Enum
</script>

```

Now you can use a CodeSmith Generator [Property directive](#) to define a property that makes use of the new type:

```

<%@ Property Name="InitialState" Type="StateEnum" Category="Options" Default="STARTED"
Description="The initial state of the Web service." %>

```

There's one more piece that you need to add to make it all work, though. Under the covers, .NET treats enumerations as integers, but you want to insert literal strings in the generated code. To make the translation, you'll also need to add a helper function inside of the script block:

```

Public Function GetState (ByVal State As StateEnum) As String
    Select Case State
        Case StateEnum.STARTED
            GetState = "STARTED"
        Case StateEnum.STOPPED
            GetState = "STOPPED"
        Case StateEnum.DISABLED
            GetState = "DISABLED"
    End Select
End Function

```

Having done this, you can get the string corresponding to the user's choice of InitialState property by inserting <%= GetState(InitialState) %> anywhere in the template. After adding enumerations, properties, and helper functions for the authentication and port properties, here's the current state of our template:

```

<%@ CodeTemplate Language="VB" TargetLanguage="T-SQL" Description="Create an HTTP Endpoint." %>
<%@ Property Name="InitialState" Type="StateEnum" Category="Options" Default="STARTED"
Description="The initial state of the Web service." %>
<%@ Property Name="Authentication" Type="AuthenticationEnum" Category="Options"
Default="INTEGRATED" Description="Authentication method." %>
<%@ Property Name="Port" Type="PortsEnum" Category="Options" Default="CLEAR" Description="Port to
use." %>

```

```

CREATE PROC dbo.PersonAddressTypeProc
AS
    SELECT
        AddressTypeID,
        Name,
        rowguid,
        ModifiedDate
    FROM
        Person.AddressType
GO
CREATE ENDPOINT GetAddressType
    STATE = <%= GetState(InitialState) %>
AS HTTP
(
    PATH = '/AddressType',
    AUTHENTICATION = (<%= GetAuthentication(Authentication) %>),
    PORTS = (<%= GetPort(Port) %>),
    SITE = 'localhost'
)
FOR SOAP
(
    WEBMETHOD 'AddressTypeList'
        (NAME='AdventureWorks.dbo.PersonAddressTypeProc'),
    BATCHES = DISABLED,
    WSDL = DEFAULT,
    DATABASE = 'AdventureWorks',
    NAMESPACE = 'http://AdventureWorks/AddressType'
)
GO

```

```

<script runat="template">
Public Enum StateEnum
    STARTED
    STOPPED
    DISABLED
End Enum

Public Enum AuthenticationEnum
    BASIC
    DIGEST
    NTLM
    KERBEROS
    INTEGRATED
End Enum

Public Enum PortsEnum
    CLEAR
    SSL
End Enum

Public Function GetState (ByVal State As StateEnum) As String
    Select Case State
        Case StateEnum.STARTED
            GetState = "STARTED"
        Case StateEnum.STOPPED
            GetState = "STOPPED"
        Case StateEnum.DISABLED
            GetState = "DISABLED"
    End Select
End Function

Public Function GetAuthentication (ByVal Authentication As AuthenticationEnum) As String
    Select Case Authentication
        Case AuthenticationEnum.BASIC
            GetAuthentication = "BASIC"
        Case AuthenticationEnum.DIGEST
            GetAuthentication = "DIGEST"
        Case AuthenticationEnum.NTLM
            GetAuthentication = "NTLM"
        Case AuthenticationEnum.KERBEROS
            GetAuthentication = "KERBEROS"
        Case AuthenticationEnum.INTEGRATED
            GetAuthentication = "INTEGRATED"
    End Select
End Function

Public Function GetPort (ByVal Port as PortsEnum) As String
    Select Case Port
        Case PortsEnum.CLEAR
            GetPort = "CLEAR"
        Case PortsEnum.SSL
            GetPort = "SSL"
    End Select
End Function
</script>

```

So far, so good. But there's still one thing missing: a connection to the database. We'll tackle that next.



Don't forget to save your template by clicking on the Save icon or selecting File -> Save from the menu located at the top of the Generator Template Editor.

Next: [Setting up a SQL Property](#)

**Setting up a SQL Property**

In order to generate code based on a database table, the template has to somehow know about the database table. This means supplying metadata through a property that refers to the table. Fortunately, CodeSmith Generator includes the SchemaExplorer library, which contains a rich set of types designed specifically for interacting with databases. One of these types, TableSchema, allows the user to pick a table from a database. You can then use the object model in the SchemaExplorer library to retrieve just about any information you need about the table and the database. Here's the [Property directive](#) that we need:

```
<@@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" Category="Context"
Description="Table that the Web service will access." %>
```

CodeSmith Generator itself doesn't have any special knowledge of the types in the SchemaExplorer library, so we need to tell it to load the assembly containing the library. It's also useful to import the SchemaExplorer namespace to keep the amount of typing we have to do to a minimum:

```
<@@ Assembly Name="SchemaExplorer" %>
<@@ Import Namespace="SchemaExplorer" %>
```

When the user selects a table with SchemaExplorer, the TableSchema object will be populated and returned to CodeSmith Generator. For the most part, this particular template can be filled out just by retrieving the names of the table, the table's owner, and the database name from this object. All of those are easily available by navigating around the [SchemaExplorer object model](#):

```
<%= SourceTable.Name %>
<%= SourceTable.Owner %>
<%= SourceTable.Database.Name %>
```

Substituting those expressions in appropriate places will get you most of the way through writing this particular template. But there's still one task left that requires a bit of coding: building the list of column names for the stored procedure.

Next: [Writing the Database Code](#)

## Writing the Database Code

The trickiest part of writing this particular template is retrieving the list of column names for the stored procedure definition. Those, too, are available from SchemaExplorer. The TableSchema object contains a Columns collection, which you can iterate through in code. You can place scripting code directly in your template by enclosing it within <% and %> tokens. Here's the code we need to build the list of columns, complete with appropriate commas:

```
<% For i As Integer = 0 To SourceTable.Columns.Count -1 %>
<%= SourceTable.Columns(i).Name %><% If i < SourceTable.Columns.Count - 1 Then %>,<% End If %>
<% Next %>
```

Note the difference here between code to execute (surrounded by <% %> tokens), expressions to evaluate (surrounded by <%= %> tokens) and static content to copy to the output (not surrounded at all). You may find keeping all this straight one of the more confusing aspects of working with CodeSmith Generator at first.

With all of the pieces in place, here's the final template:

```
<@@ CodeTemplate Language="VB" TargetLanguage="T-SQL" Debug="True" Description="Create an HTTP
Endpoint." %>
<@@ Property Name="InitialState" Type="StateEnum" Category="Options" Default="STARTED"
Description="The initial state of the Web service." %>
<@@ Property Name="Authentication" Type="AuthenticationEnum" Category="Options"
Default="INTEGRATED" Description="Authentication method." %>
<@@ Property Name="Port" Type="PortsEnum" Category="Options" Default="CLEAR" Description="Port to
use." %>
<@@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" Category="Context"
Description="Table that the Web service will access." %>
<@@ Assembly Name="SchemaExplorer" %>
<@@ Import Namespace="SchemaExplorer" %>
```



```

CREATE PROC dbo.<%= SourceTable.Owner %><%= SourceTable.Name %>Proc
AS
    SELECT
        <% For i As Integer = 0 To SourceTable.Columns.Count -1 %>
        <%= SourceTable.Columns(i).Name %><% If i < SourceTable.Columns.Count - 1 Then %>,<% End If
%>
        <% Next %>
    FROM
        <%= SourceTable.Name %>
GO
CREATE ENDPOINT Get<%= SourceTable.Name %>
    STATE = <%= GetState(InitialState) %>
AS HTTP
(
    PATH = '/<%= SourceTable.Name %>',
    AUTHENTICATION = (<%= GetAuthentication(Authentication) %>),
    PORTS = (<%= GetPort(Port) %>),
    SITE = 'localhost'
)
FOR SOAP
(
    WEBMETHOD '<%= SourceTable.Name %>List'
        (NAME='<%= SourceTable.Database.Name %>.dbo.<%= SourceTable.Owner %><%= SourceTable.Name
%>Proc'),
        BATCHES = DISABLED,
        WSDL = DEFAULT,
        DATABASE = '<%= SourceTable.Database.Name %>',
        NAMESPACE = 'http://<%= SourceTable.Database.Name %>/<%= SourceTable.Name %>'
    )
)
GO

```

```

<script runat="template">
Public Enum StateEnum
    STARTED
    STOPPED
    DISABLED
End Enum
Public Enum AuthenticationEnum
    BASIC
    DIGEST
    NTLM
    KERBEROS
    INTEGRATED
End Enum
Public Enum PortsEnum
    CLEAR
    SSL
End Enum

Public Function GetState (ByVal State As StateEnum) As String
    Select Case State
        Case StateEnum.STARTED
            GetState = "STARTED"
        Case StateEnum.STOPPED
            GetState = "STOPPED"
        Case StateEnum.DISABLED
            GetState = "DISABLED"
    End Select
End Function

Public Function GetAuthentication (ByVal Authentication As AuthenticationEnum) As String
    Select Case Authentication
        Case AuthenticationEnum.BASIC
            GetAuthentication = "BASIC"
        Case AuthenticationEnum.DIGEST
            GetAuthentication = "DIGEST"
        Case AuthenticationEnum.NTLM
            GetAuthentication = "NTLM"
        Case AuthenticationEnum.KERBEROS
            GetAuthentication = "KERBEROS"
        Case AuthenticationEnum.INTEGRATED
            GetAuthentication = "INTEGRATED"
    End Select
End Function

Public Function GetPort (ByVal Port as PortsEnum) As String
    Select Case Port
        Case PortsEnum.CLEAR
            GetPort = "CLEAR"
        Case PortsEnum.SSL
            GetPort = "SSL"
    End Select
End Function
</script>

```



Click [here](#) to download the completed template.

Next: [Testing the Final Result](#)

More Information:

[SchemaExplorer](#)

## Testing the Final Result

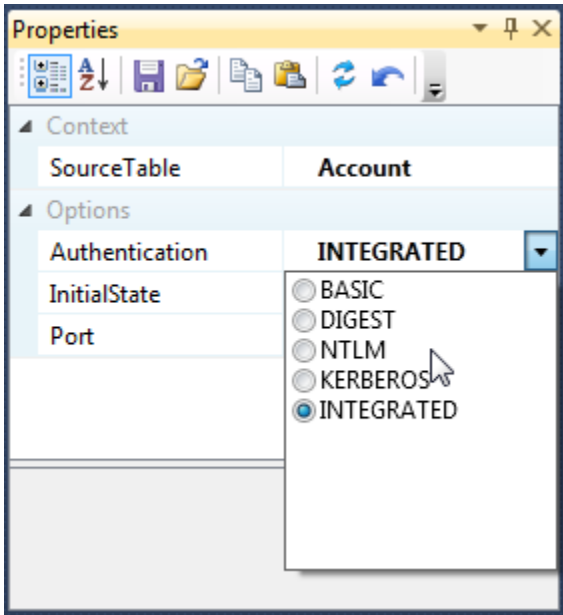
Now that you've written the template, it's easy to test it out. First you'll need to compile the template, so that CodeSmith Generator will display the

right properties in the Properties Window. Click the Build button on the toolbar or press Ctrl+Shift+B to do this. Assuming that there are no errors in the template, you'll see progress messages in the Output Window:

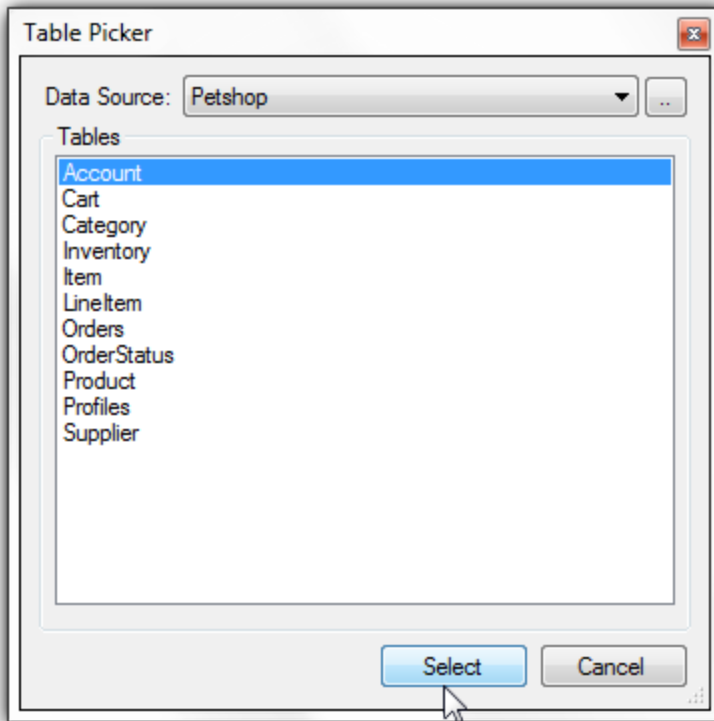
```
----- Compile started -----  
Build complete -- 0 errors, 0 warnings  
----- Done -----  
Compile succeeded
```

✔ If there are any errors in the template, they'll display in the Output Window.

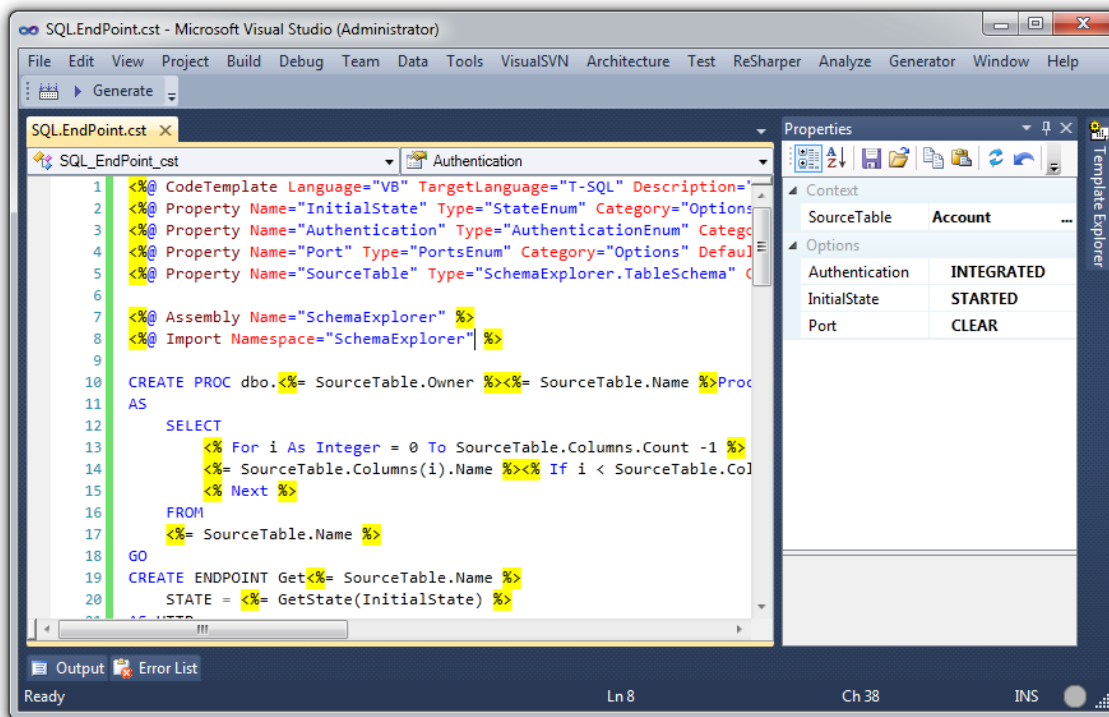
Now you can use the Properties Window to enter values for the template's properties. For the three enumerated properties, you'll find that CodeSmith Generator provides dropdown lists to choose from.



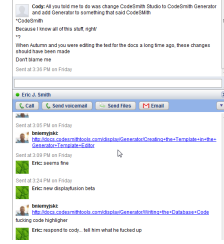
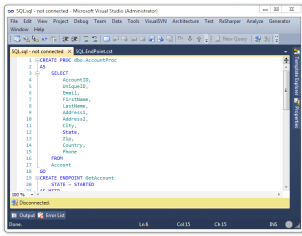
To set the SourceTable property, click in the property value. CodeSmith Generator will display a builder button with three dots. Click the builder button to open the Table Picker dialog box.



Here you can select the data source, and the table within that data source, to use with the template. You can also click the builder button next to the Data Source combo box to create new data sources. After choosing a table, click the Select button to return to the editor.



When you're done setting properties, click the Generate toolbar button or press F5 to run the template. CodeSmith Generator will generate the template's output and switch to the Output tab so that you can save or copy the output.



More Information:

[SchemaExplorer](#)

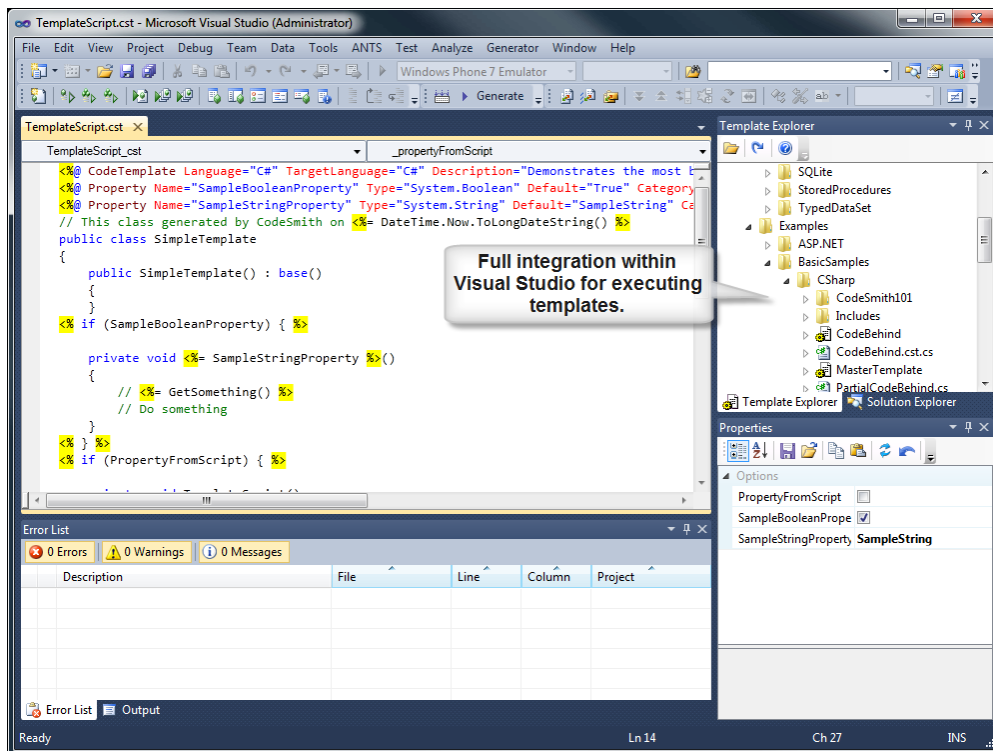
[Building, Running, and Compiling Templates](#)

## Visual Studio Integration

In addition to the standalone user interface [Template Explorer](#), CodeSmith Generator also offers integration with Microsoft Visual Studio. This integration takes many forms:

### Template Explorer

To use [Template Explorer](#) from within Visual Studio, select Template Explorer from the Generator menu which is located between the Tools and Help menu. This will open the Template Explorer tool window. This window can be floating or docked, just like any other Visual Studio tool window.



Template Explorer has the same functionality in Visual Studio that it does as a standalone program.

### Template Editor

The [Generator Template Editor](#) is now integrated seamlessly into Visual Studio (as pictured above). The editor features rich IntelliSense, auto completion, documentation capabilities, go-to-definition support and much more.

```

21     <% MyMethod(StringProperty); %>
22 }
23
24 <script runat="template">
25
26     </// <summary>
27     </// My String Property
28     </// </summary>
29     public System.String StringProperty { get; set; }

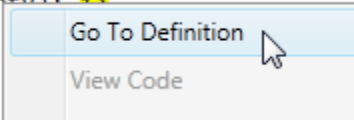
```

The Template Editor features rich IntelliSense and auto completion support just like Visual Studio. This allows you to increase productivity while developing templates.

```

<% MyMethod(StringProperty); %>
}
<script runat="template">

```



The Template Editor also features Go-To-Definition support which will take you instantly to the identifier's definition via the Object Browser.

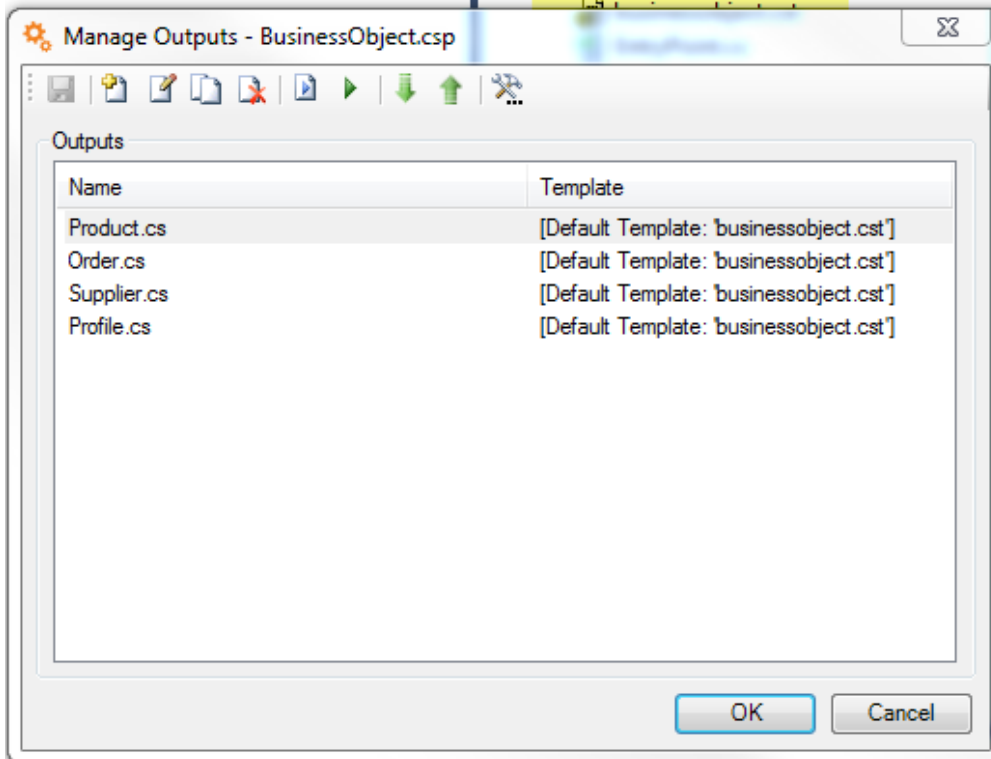
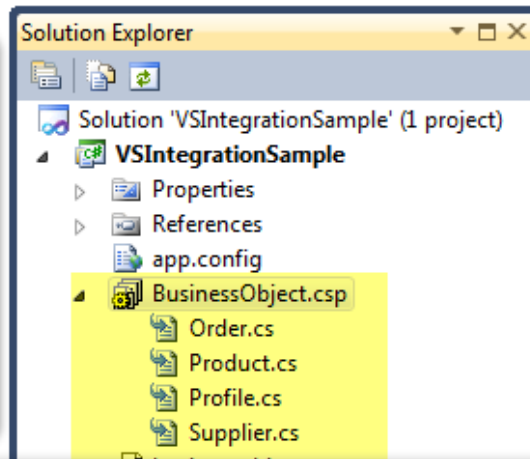
## CodeSmith Generator Project Integration

[CodeSmith Generator Projects](#) can be used to generate code within your Visual Studio Projects.

### Visual Studio Project Integration:

Have an integrated CodeSmith Project control multiple outputs from a single source template.

**Any outputs can automatically be added to your Visual Studio Projects!**



### ActiveSnippets

[ActiveSnippets](#) can be used to generate snippets of code similar to Visual Studio's snippets except with the full power of CodeSmith Generator templates and the ability to use complex metadata like database schema and XML.

```

using System;
using System.Collections.Generic;
using System.Text;

namespace CsharpCodeGeneratorSample
{
    class Sample
    {
        tp Petshop.dbo.Orders
    }
}

```

### Fantastic Snippet Power

You can use an alias for your ActiveSnippet. Arguments can be simple and even complex objects such as a table or xml data as a template argument. Type CTRL-E + CTRL-E to expand the ActiveSnippet.

You can easily name your snippets to quickly generate exactly what you want.

```

using System;
using System.Collections.Generic;
using System.Text;

namespace CsharpCodeGeneratorSample
{
    class Sample
    {
        private int _orderId;

        public int OrderId
        {
            get { return _orderId; }
            set { _orderId = value; }
        }

        private string _userId;

        public string UserId
        {
            get { return _userId; }
            set { _userId = value; }
        }

        private System.DateTime _orderDate;

        public System.DateTime OrderDate
        {
            get { return _orderDate; }
            set { _orderDate = value; }
        }
    }
}

```

### Expanding Active Snippets: Expand: CTRL-E + CTRL-E

Once expanded you now see the power of CodeSmith Templates as an Active Snippet and increase your developer productivity exponentially.

Running the snippet is very easy, all you need to do is press CTRL-E twice or select Generator -> Expand ActiveSnippet from the Visual Studio menu.

**Learn More**



You can also check out this video tutorial to learn more!

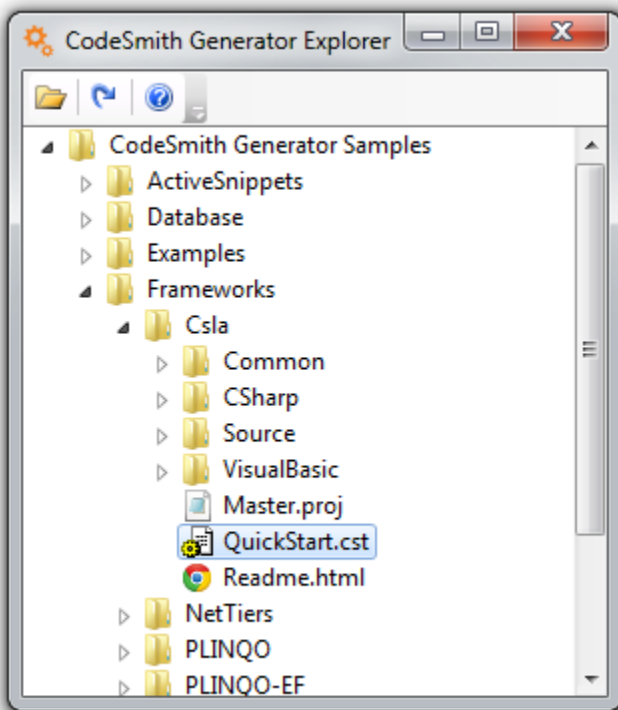
## Using Template Explorer

Using Template Explorer covers the below sections:

- [What is Template Explorer?](#)
- [The Template Explorer Toolbar](#)
- [Managing the Folder Tree](#)
- [Editing Templates](#)
- [Executing Templates](#)
- [Working with the Output Window](#)

## What is Template Explorer?

The Template Explorer, also known as CodeSmith Generator Explorer, provides an easy interface for organizing and executing CodeSmith Generator templates. Just as Windows Explorer sorts your files into folders, Template Explorer sorts templates into folders to make it easier to find the templates that you want to work with.



## The Template Explorer Toolbar

The Template Explorer Toolbar includes the following buttons:



**Create a template folder shortcut:** Clicking this button will open a Browse for Folder dialog box that lets you select any existing folder on your computer. Click OK to add a shortcut to the selected folder as a top-level node in Template Explorer. When you install CodeSmith Generator, Template Explorer is pre-populated with a single shortcut to the [Sample Templates](#) folder in your installation.



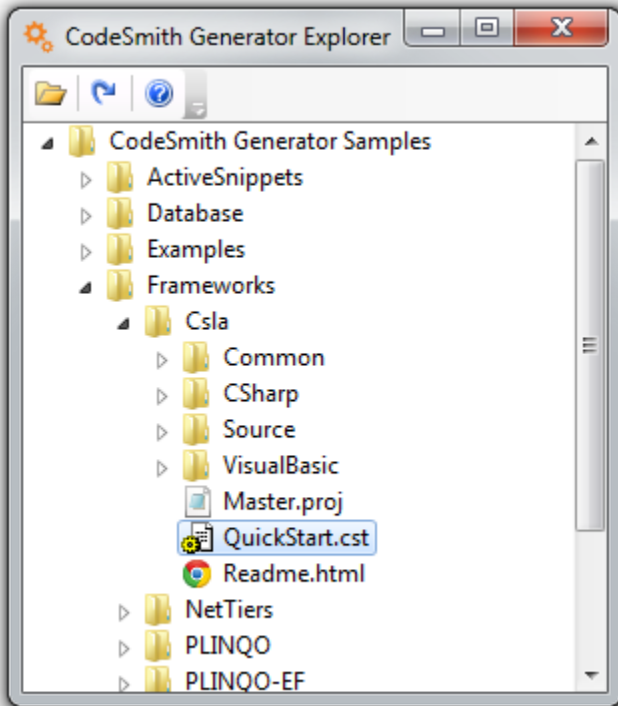
**Toggle Top Most Window Mode:** By default, CodeSmith Generator Explorer (which hosts Template Explorer) behaves as a normal window that can be overlaid by other windows. If you select this button, Template Explorer will float on top of all other windows, remaining permanently visible even if another window that would otherwise hide it has the focus. This is especially useful when you want to generate code by dragging and dropping templates from Template Explorer to any application that supports dropping text.



**About CodeSmith:** This button displays the version and licensing information for your copy of CodeSmith Generator.

## Managing the Folder Tree

Folders may contain subfolders, templates, or both. At any time, one node in the folder tree will be selected. The selected node is indicated by a highlight. In the screenshot below, the CSLA QuickStart template is selected. You can select a node by clicking on it with the mouse.



**i** You can also move the selection by using the up or down arrow keys, or by typing the first letter of the name of the node.

A folder that can be expanded as indicated by a arrow sign to its left. To expand a folder, click the plus sign, or double-click the folder or its name, or select the folder and click the right arrow or the plus sign on the numeric keypad. A folder that can be collapsed is indicated by a darkened arrow sign to its left. To collapse a folder, click the arrow sign, or double-click the folder or its name, or select the folder and click the left arrow or the minus sign on the numeric keypad.

You can perform a variety of other operations from the CodeSmith Generator Explorer context menus. These menus differ depending on which node you right-click on.

**i** All of your existing Windows Explorer context menu items (E.G., Delete, Rename...) will also be displayed in these context menu's.

**i** You can open a template folder in Windows Explorer by right-clicking on a folder and selecting the Open context menu item.

### **Context Menu for a Folder**

Right-clicking on a folder will bring up a shortcut menu with the following choices:

- **New**
  - **CodeSmith Generator Template (CSharp):** Creates a new template using CSharp as its code-behind language.
  - **CodeSmith Generator Template (Visual Basic):** Creates a new template using Visual Basic as its code-behind language.
  - **CodeSmith Generator Project:** Creates a new [CodeSmith Generator Project](#) file.
  - **CodeSmith Generator map:** Creates a new blank [CodeSmith Generator Map](#) file.
  - **File:** Creates a new empty file. By default the file will have a .txt extension.
  - **Folder:** Creates a new sub-folder.

If you are right-clicking on a template folder shortcut (top-level folder in the tree) the following shortcut menu items will also be displayed:

- **Remove Shortcut.** Deletes this folder shortcut from CodeSmith Explorer. This does not delete the underlying files from your hard drive.

- **Include Subfolders.** When checked, CodeSmith Generator Explorer will also display subfolders of the target folder. Otherwise, it will only display templates in the target folder of the shortcut itself.

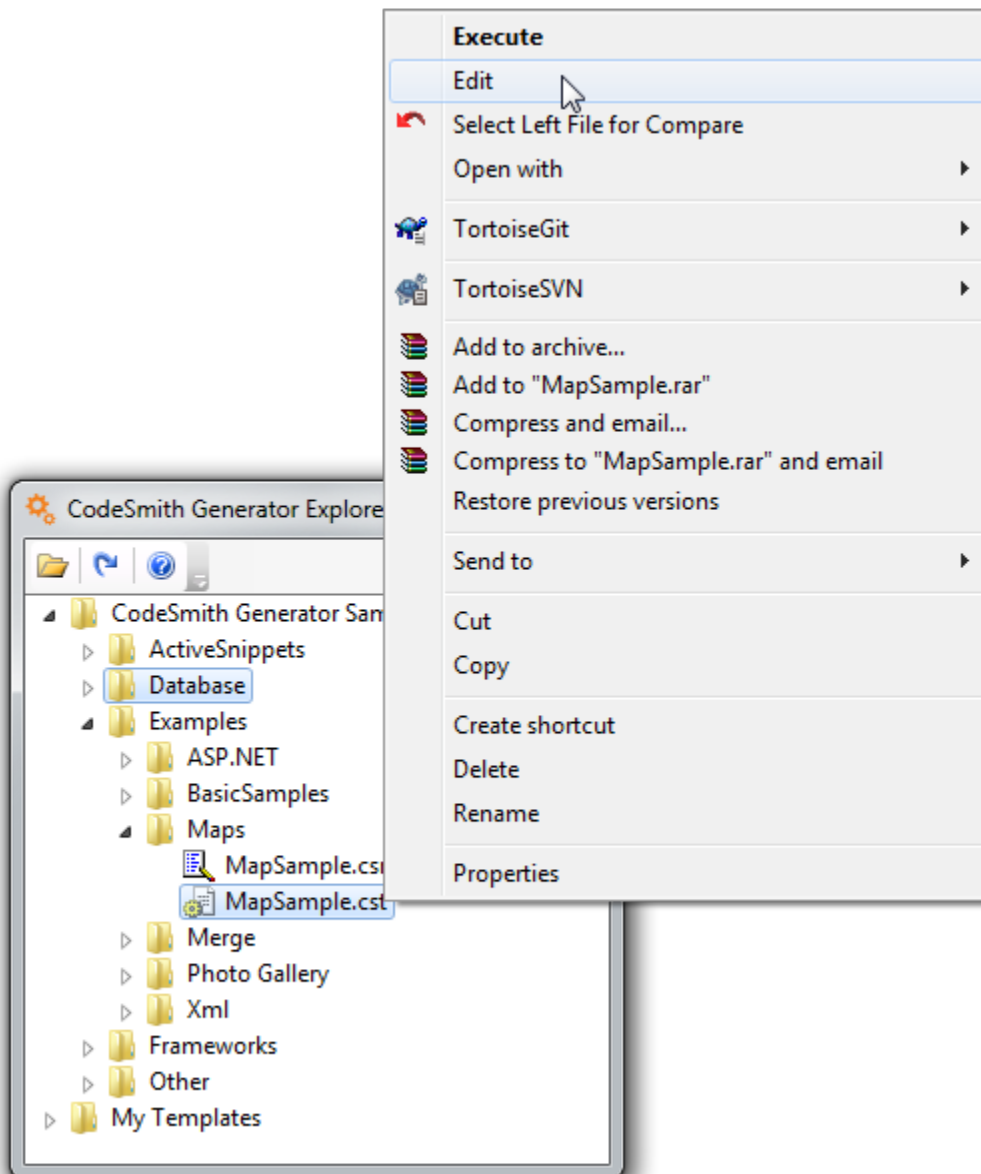
### Context Menu for a Template

Right-clicking on a template brings up a shortcut menu with the following choices:

- **Execute.** Execute the template.
- **Edit.** Open the template for editing.

## Editing Templates

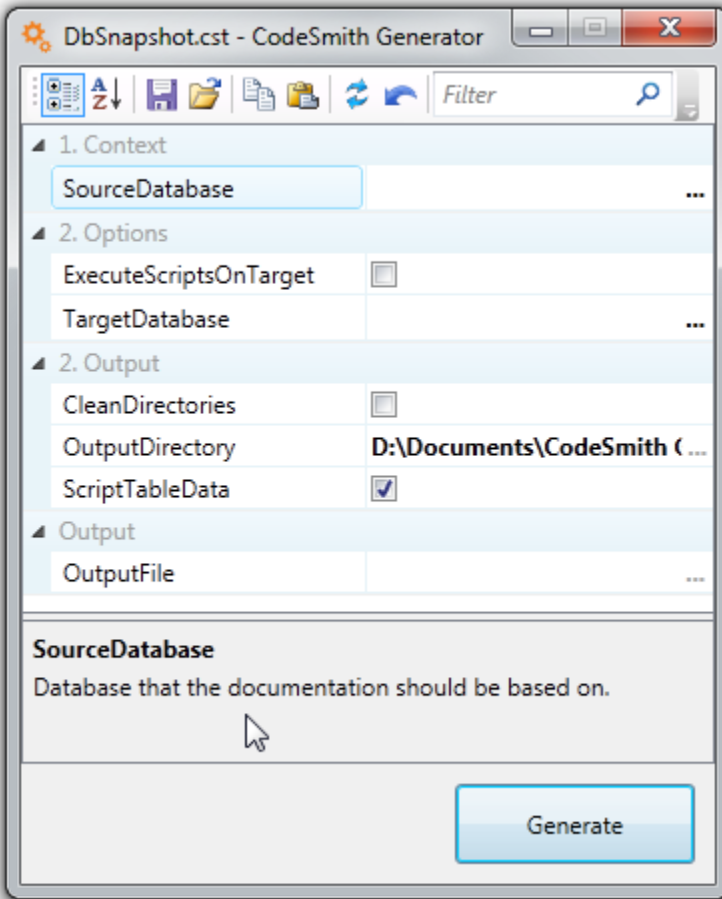
To edit a template, right-click the template in Template Explorer or Window Explorer and select Edit.



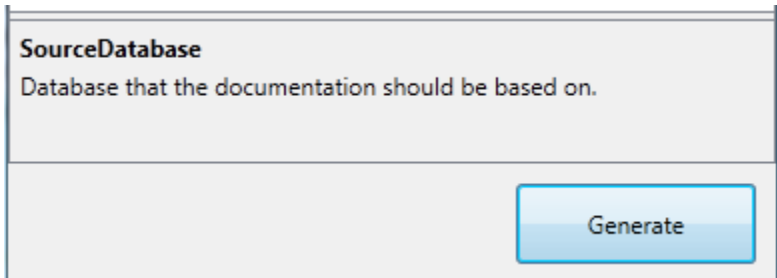
This will open the template in the default template editor. Template Explorer will use an existing editor session if possible, otherwise it will launch a new editor session.

## Executing Templates

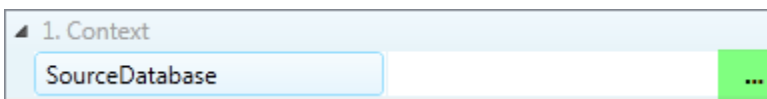
To execute a template from Template Explorer, double-click the template, or right-click the template and select Execute. You can also drag the template from Template Explorer and drop it on any application that supports drag-and-drop to generate code at the location where you dropped the template. Any of these actions will open the template's property sheet.



The template's property sheet shows you all of the properties that you can set for this template. Properties can be required or optional. You need to supply values for all required properties before CodeSmith Generator can generate the code for you. Depending on how a property is defined in the template, you may be able to type in an arbitrary value, select a value from a predefined list, or choose a value by navigating to a dialog box from a builder button within the property sheet. Properties may also have default values. As you select each property, a description will appear at the bottom of the property sheet to tell you more about that property.




In the screenshot above, the user has selected the SourceDatabase property, and the description indicates that this property specifies the Database that the documentation will be created for.



With the SourceDatabase property selected, the right side of the property sheet shows the builder button (with three dots) as shown highlighted in green. Clicking this button will open a separate dialog box (in this case, a dialog box supplied by CodeSmith Generator's own [SchemaExplorer](#) metadata extension) to help you pick a value for this property.

When you have finished setting properties for the template, you're ready to generate code. To do this, click the Generate button at the bottom of

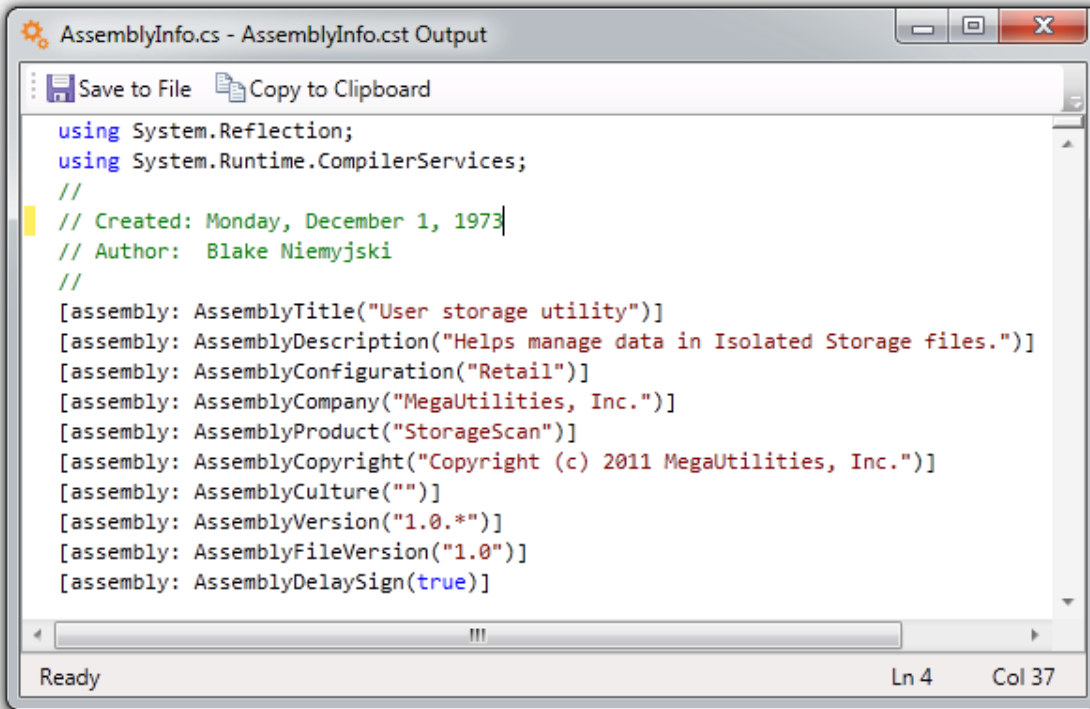
the template's property sheet. CodeSmith Generator will take the property values that you entered and combine them with the template to create the code, and display it in an [Output Window](#) or output it to a specific directory. In this case the code will be generated to the folder specified in the `OutputDirectory` property.

 [Click here to learn about the property sheet toolbar.](#)

Advanced: [Using a CodeSmith Generator Project to Execute CodeSmith Templates from Anywhere](#)

## Working with the Output Window

Template Explorer will automatically display an Output Window when you click the Generate button. The Output Windows' contents can be modified at any time which allows you to make changes anytime to the document.



```
using System.Reflection;
using System.Runtime.CompilerServices;
//
// Created: Monday, December 1, 1973
// Author: Blake Niemyjski
//
[assembly: AssemblyTitle("User storage utility")]
[assembly: AssemblyDescription("Helps manage data in Isolated Storage files.")]
[assembly: AssemblyConfiguration("Retail")]
[assembly: AssemblyCompany("MegaUtilities, Inc.")]
[assembly: AssemblyProduct("StorageScan")]
[assembly: AssemblyCopyright("Copyright (c) 2011 MegaUtilities, Inc.")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyFileVersion("1.0")]
[assembly: AssemblyDelaySign(true)]
```

The output panel has its own toolbar with two buttons.



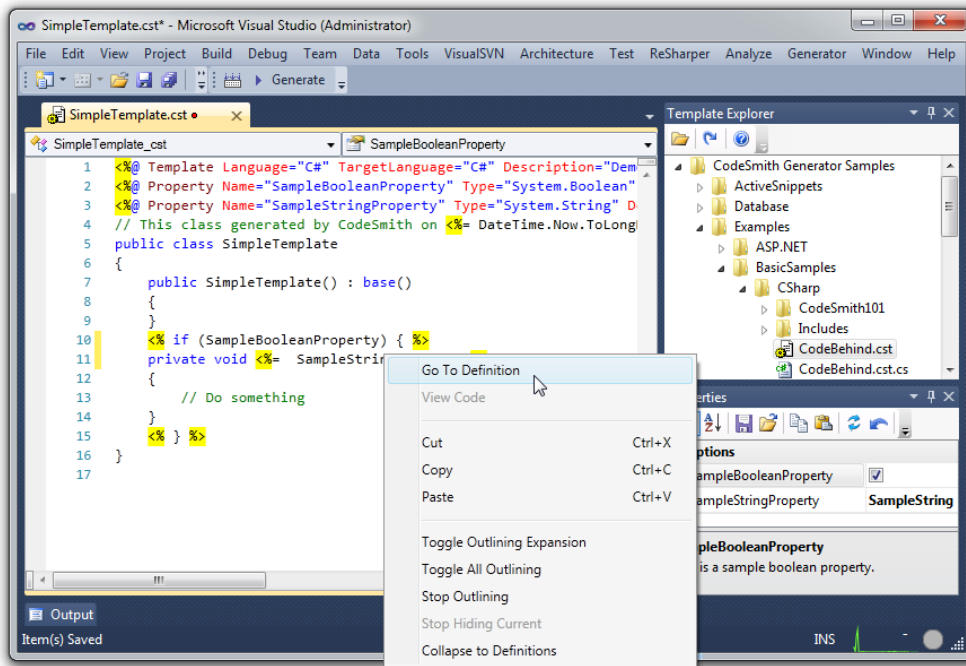
**Copy to Clipboard:** Copies the current document to the Windows clipboard.



**Save to File:** Opens a Save As dialog box to let you save the current document to a new file on your hard drive.

## Using the Template Editor

The Generator Template Editor provides a complete integrated development environment (IDE) for CodeSmith Generator templates. You can use Generator Template Editor to edit, compile, and run CodeSmith Generator templates. The Generator Template Editor includes features designed to make building and debugging templates easier.



The Template Editor offers a superset of the functionality of [Template Explorer](#). When you just want to execute templates and generate code, you'll find it's faster to use [Template Explorer](#) to get your work done. But when you have templates under active development, [Template Editor](#) should be your tool of choice.

This section will cover the following topics:

- [Template Editor User Interface](#)
- [Template Editor Features](#)
- [Building, Running, and Compiling Templates](#)
- [Customizing CodeSmith Generator](#)

## Template Editor User Interface

The Generator Template Editor user interface includes a number of separate elements as shown below, each with its own purpose. The following document will step you through the different elements.

### User interface elements

The [Template Windows](#)([Template Document](#), [Generated Document](#)), [Template Explorer](#), [Properties Window](#), and [Output Window](#) can be docked to any side of the template documents (shown in green below) or floating. Docked windows can also be set to auto-hide by clicking on the pin button in the upper right hand side of the window. When you hover the mouse over an auto-hide window, it "slides out" to become fully visible, covering other user interface elements. Multiple floating windows can be docked to one another.

### Generator menu

The [Generator menu](#) is located at the top of the [Template Editor](#) and is highlighted in light blue. This menu allows you to display additional user interface elements like [Schema Explorer](#), [Template Explorer](#), [Map Editor](#), [manage data sources](#), [ActiveSnippet configuration](#), [about dialog](#), [error window](#) and much more.

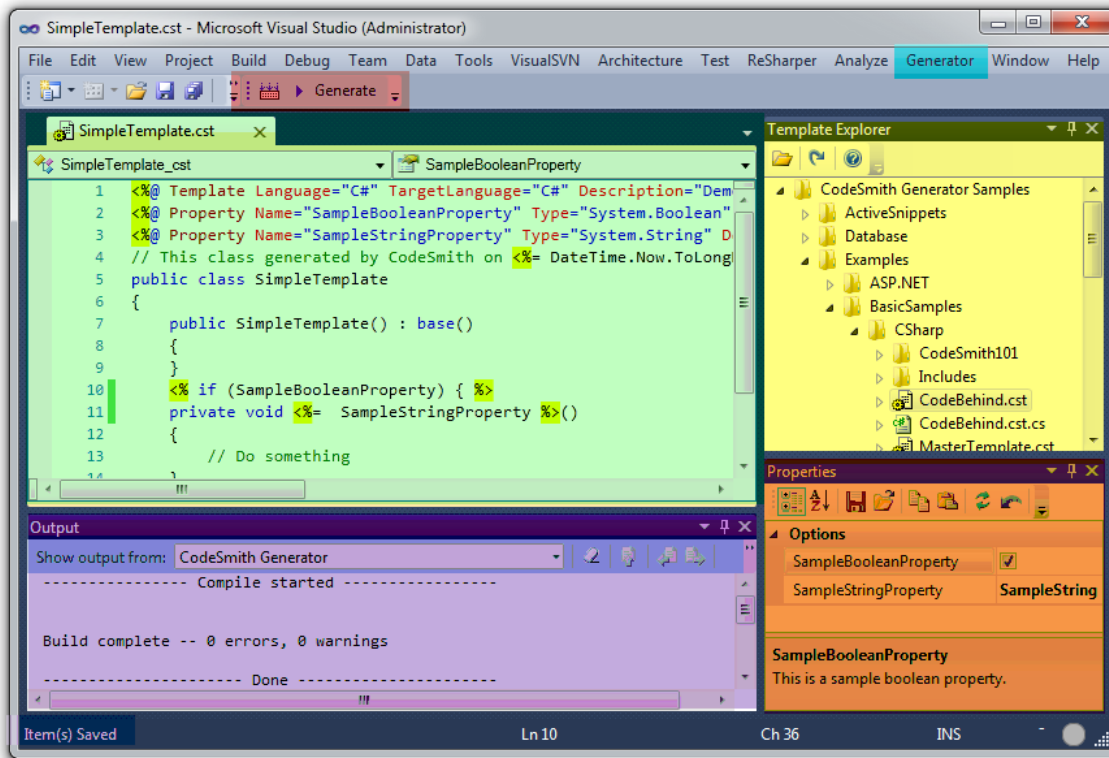
### Template Editor toolbar

The [Template Editor toolbar](#) is located at the top of the [Template Editor](#) and is highlighted in red. This toolbar allows you to quickly generate or build a template.

### Template Documents

[Template documents](#) refer to a document window that consists of a specific function. Here is a general overview of the various template document types:

- **Template document:** A template document gives you the ability to edit a template. A template document is highlighted in green below. Click [here](#) to learn more about the template document.
- **Generated document:** A generated document shows you the generated template content. Click [here](#) to learn more about the generated document.



### Template Explorer

Template Explorer is deeply integrated into the Template Editor allowing for quick access to all of your existing templates. You can double click on a template to edit the template in the Template Editor and much more. The Template Explorer is highlighted above in yellow.

### Properties window

The [properties window](#) lets you view and edit the property settings for the current template, similar to the property sheet you can get by invoking Execute from Template Explorer. The properties window is highlighted above in orange.

### Output window

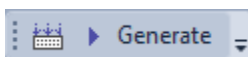
The [output window](#) is used by CodeSmith Generator to send status messages to you. The properties window is highlighted above in purple.


### Error window

The [error window](#) is used by CodeSmith Generator to display template document errors or warnings that occurred while editing or compiling.

## Template Editor Toolbar

The Template Editor Toolbar is located at the top of the Template Editor and can be seen below.




 This toolbar is only shown when a Template Document has focus.

### Toolbar Actions

The toolbar contains the following actions:




**Build:** Clicking this button compiles the current template. The [Output window](#) will show the results of the build operation.

 You can also build a template by pressing F6.

### Generate

**Generate:** Clicking this button executes the current template, using values from the [Properties window](#) together with the template to create the generated code. The generated code will be displayed in the [Output tab](#) once this operation is complete.

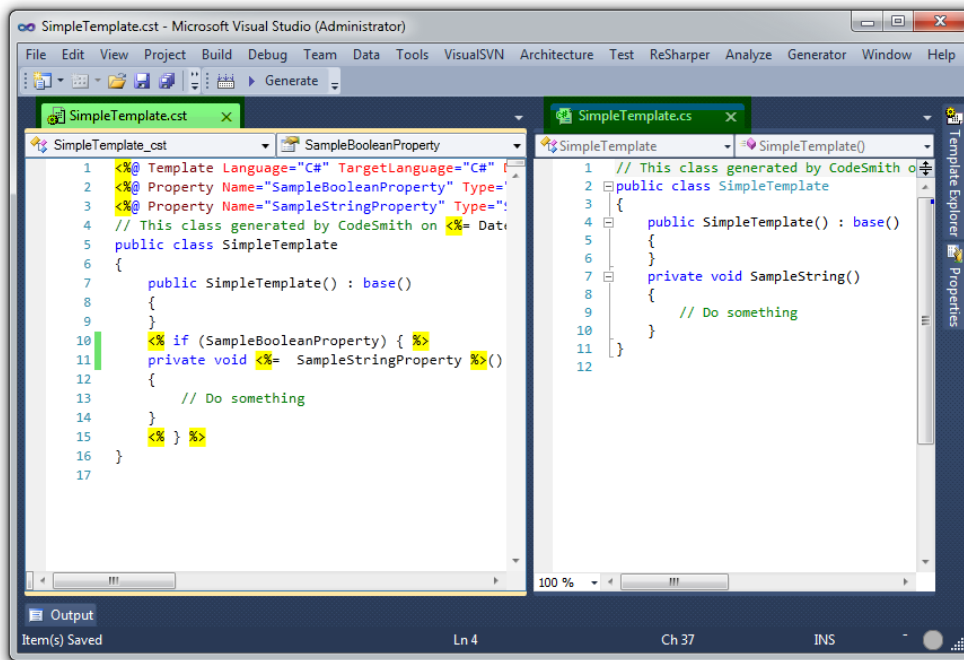
 You can also build a template by pressing F5.

## Template Documents

Template documents refer to a document window that consists of a specific function. Template documents allow you to edit CodeSmith Generator templates, as well as to view the generated code from templates. Here is a general overview of the various template document types:

- **Template document:** A template document gives you the ability to edit a template. A template document is highlighted in green below. Click [here](#) to learn more about the template document.
- **Generated document:** A generated document shows you the generated template content. Click [here](#) to learn more about the generated document.

You can have multiple documents open at the same time in the Template Editor. You can switch between documents by clicking on a documents tab. A documents tab is located at the top of a documents' design surface as highlighted in green below.



In the screenshot above, a template document is displayed on the left and a generated document is shown on the right.

### Template Document

The template document is the editing surface of the Template Editor. A template document consists of the template source code.



```

2 | <%@ Property Name="SampleBooleanProperty" Type="System.Boolean"
3 | <%@ Property Name="SampleStringProperty" Type="System.String"
4 | // This class generated by CodeSmith Generator on <%= DateTime.N
5 | public class SimpleTemplate
6 | {
7 |     public SimpleTemplate() : base()
8 |     {
9 |     }
10 | <% if (SampleBooleanProperty) { %>
11 | |
12 | private void <%= SampleStringProperty %>()
13 | {
14 |     // Do something
15 | }
16 | <% } %>
17 | }
18 |

```


### Generated Document

The generated document displays the generated code produced by combining the template with the property values entered by the user in the Properties window. The generated document will be displayed after you generate a template.

```


1 | // This class generated by CodeSmith Generator
2 | public class SimpleTemplate
3 | {
4 |     public SimpleTemplate() : base()
5 |     {
6 |     }
7 |     private void SampleString()
8 |     {
9 |         // Do something
10 |    }
11 | }

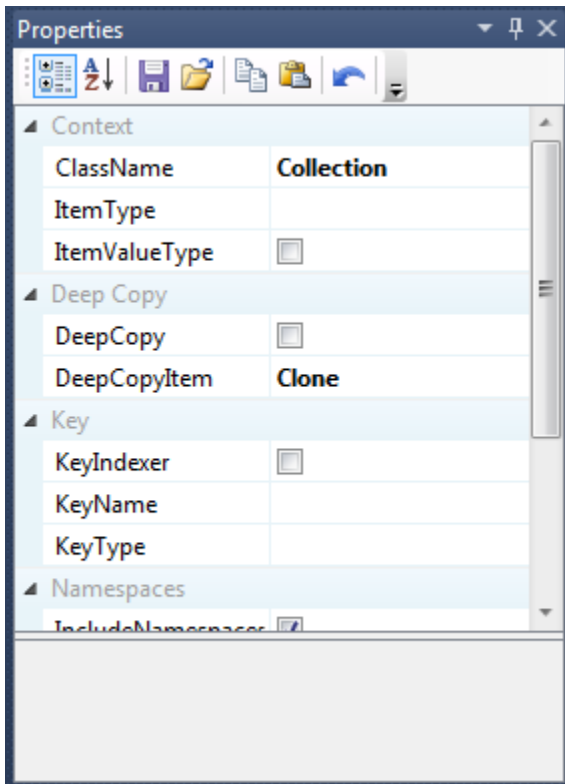
```

 The generated document will not be shown if the templates OutputType CodeTemplate Directive attribute value is set to None.

### The Properties Window


The Properties window lets you view and edit the property settings for the current template. It is similar to the property sheet you can get by invoking Execute from Template Explorer.


 You can navigate to the Properties window by selecting Properties Window from the View menu or by pressing F4.



Before you can generate code from a template, you must supply values for all of the non-optional properties in the Properties window.

One of the features of CodeSmith Generator is the ability to cache property set values for each template you open. This feature gives the ability to persist property set values from one Generator session to the next for that particular template. This includes when you change a template, recompile a template, close and re-open Generator.

 You can toggle the availability of **Property Persistence** by using the Enable Property Persistence checkbox in the [Options](#) dialog.

 [Click here](#) to learn how to set property values.

### **Property Sheet Toolbar**

The Template Explorer property sheet toolbar includes the following buttons:



**Categorize:** By default, the properties in the property sheet are sorted by categories. If you'd prefer them in a single alphabetical list, click the Alphabetic toolbar button.



**Alphabetical:** Sorts the properties in the property sheet alphabetically by property name.



**Save Property Set:** Opens a Save As dialog box to let you save the settings from the property sheet as a [CodeSmith Generator Project](#) file.



**Open Property Set:** Opens a File Open dialog box to let you select an existing [CodeSmith Generator Project](#) file. The settings from the XML file will be loaded into the property sheet.



**Copy Property Set:** Copies the settings from the property sheet into a [CodeSmith Generator Project](#) and to the Windows clipboard as an

XML file. CodeSmith Project files can be used as input to the [CodeSmith Generator Console application](#).



**Paste Property Set:** Restores the property values from the [CodeSmith Generator Project file](#) (the Xml file from Copy Property Set) stored in the Windows clipboard.



**Refresh Property Values:** Refreshes the values of the property sheet and also clears any cached values like cached database information.




**Reset Properties to Default:** Resets the property values to the default values.



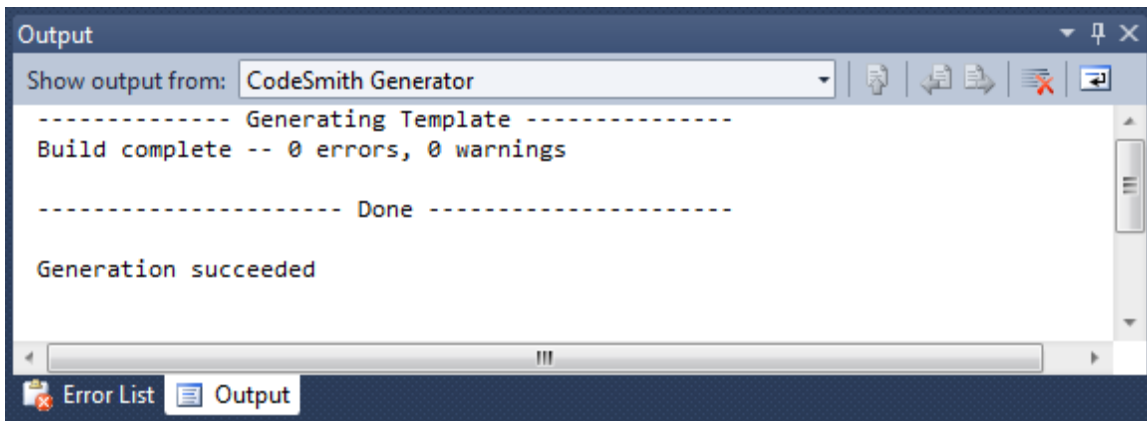
**Filter:** You can also Filter the properties that are displayed in the property sheet by typing in the name of the property or properties you are looking for. For example, if you are looking for a property that starts with 'a' or contains the letter 'a', just type 'a' into this filter box.



To clear any cached database schema information press the  Refresh Property Values button at the top of the property sheet.

## The Output Window

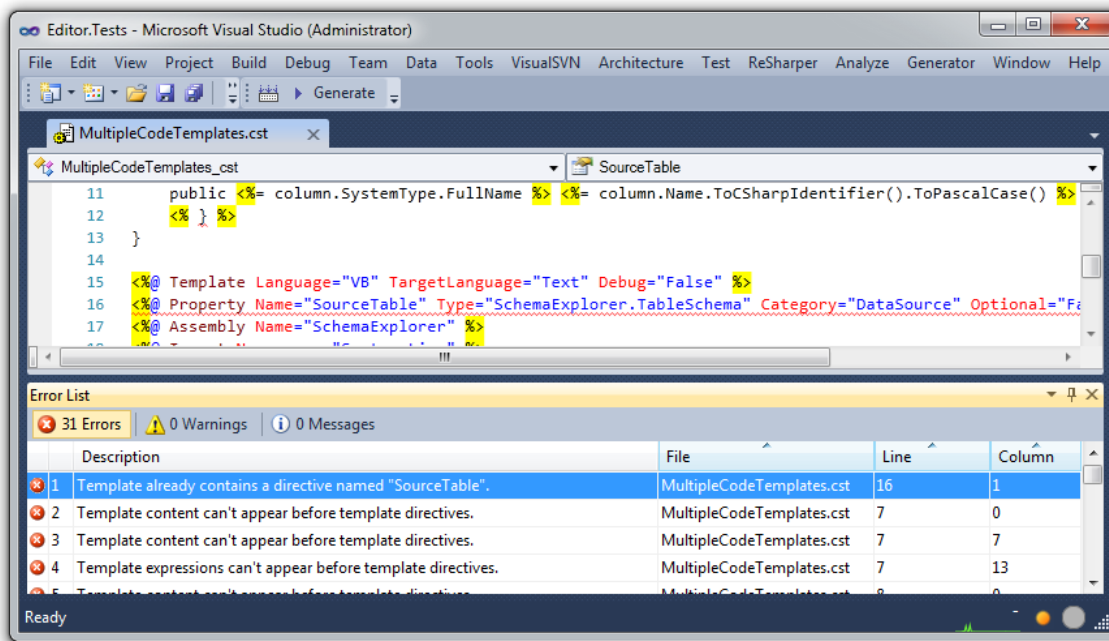
The Output window is used by CodeSmith Generator to send status messages to you.



The combo box at the top of the Output window can be used to switch between the Build pane and the Debug pane.

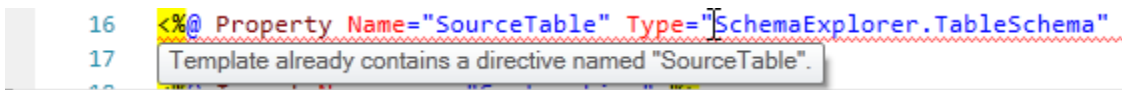
## The Error Window

The error window allows you to view any template document errors or warnings that occurred while editing or compiling.



The image above contains a template document and the error window. The error window contains errors for the MultipleCodeTemplates.cst template document. The error window can show errors, warnings or messages.

On the first line of the error window, there is an error for a duplicate property directive declaration. The line specifies the file that the error resides in as well as the line and column of the error. You can double-click any where on this line to jump you to the place of the error.



The Template Editor also helps you identify errors by placing a red squiggles under each error. You can hover over the squiggle to bring up the error information as shown above.

## Template Editor Features

Template Editor is a modern, full-featured IDE with advanced capabilities that rival those of Visual Studio .NET. Features of the editor include:

- Themes and Syntax Highlighting
- Template Navigation
- Bracket highlighting
- Documentation Comment Editing
- Find and Replace
- Incremental search
- Keyboard shortcuts
- Line Modification Markers
- Outlining
- Statement completion
- Tab groups

## Bracket Highlighting


When the cursor is in front of a bracket, the editor will outline both that bracket and the corresponding open or close bracket:

```

internal Hashtable InnerHash
{
    get
    {
        return innerHash;
    }
}

```

This works with parentheses, square brackets, and curly braces.

 You can move the cursor quickly to the corresponding bracket by typing Ctrl+].

## Documentation Comment Editing

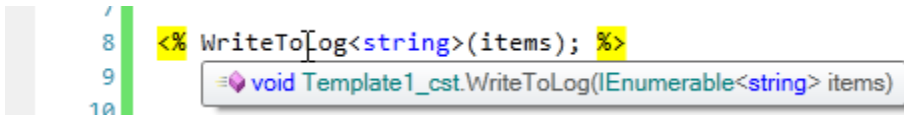
The Template Editor allows you to easily document your template code. As an added benefit, the documentation you provide will show up in [Statement Completion and Quick Info](#). This will allow you to quickly see what a method, property, or argument does without inspecting the code, therefore saving you valuable time.

### Example

In this example we are going to assume that we are editing an existing template that already has some methods defined. We will come across a method that we don't know what its responsibility is. We will then investigate and add some documentation for future reference.

#### Inspecting the code

We are inspecting existing code and have no idea what the `WriteToLog<T>` method does (as shown in the image below). We decide to hover over the method to see any comments via [Quick Info](#).



```
8 <% WriteToLog<string>(items); %>
9
10
```

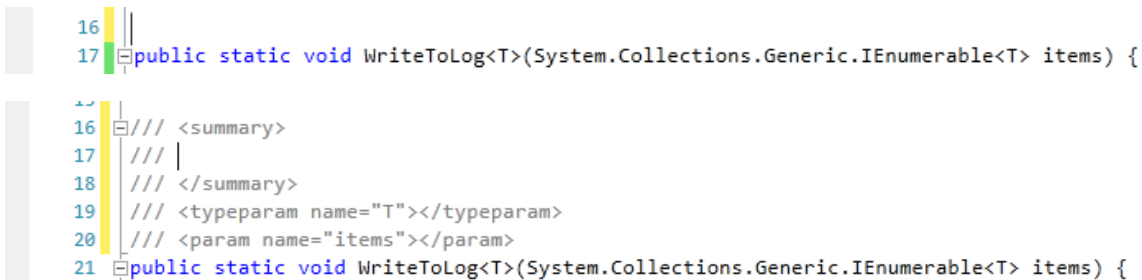
void Template1\_cst.WriteToLog(IEnumerable<string> items)

With the lack of documentation we have no idea what the method does. So we do a Go To Definition on `WriteToLog` and inspect the code.

#### Adding documentation


Now that we know what the method does, let's add some documentation so anyone using the templates in the future will know what this method does.

When typing `///` (CSharp) or `'''` (Visual Basic) immediately before a type or member, stub documentation comments will automatically be inserted.

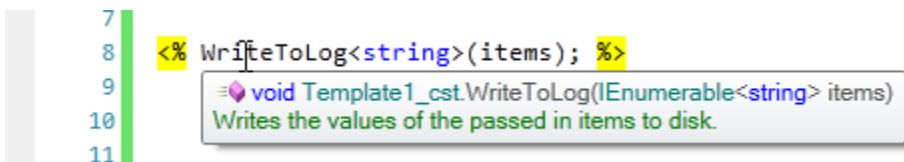


```
16
17 public static void WriteToLog<T>(System.Collections.Generic.IEnumerable<T> items) {
18
19     /// <summary>
20     /// |
21     /// </summary>
22     /// <typeparam name="T"></typeparam>
23     /// <param name="items"></param>
24     public static void WriteToLog<T>(System.Collections.Generic.IEnumerable<T> items) {
```

We will now fill out the summary section of the documentation, so others will know what this method does.

 As Enter is pressed within a documentation comment, `///` (C#) or `'''` (VB) are auto-inserted on the next line for continuation of the comment.

Now, when we hover over the `WriteToLog<T>` method we are prompted with a description!



```
7
8 <% WriteToLog<string>(items); %>
9
10
11
```

void Template1\_cst.WriteToLog(IEnumerable<string> items)  
Writes the values of the passed in items to disk.

## Find and Replace

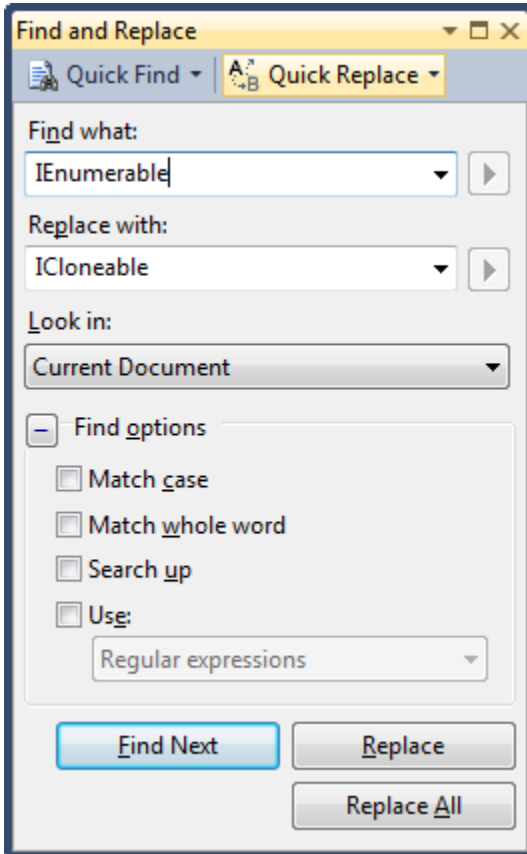
The Find and Replace dialog allows you to quickly find and replace text inside of a template.

## Opening the Find and Replace dialog

You can open the Find and Replace dialog by pressing Ctrl+F or Ctrl+H.

## Find and Replace Options

The following options are available for



**Find what:** Enter the text that you wish to find.

**Replace with:** Enter the text that you wish to replace the found text.

**Look in:** Allows you to configure where you want to search. To search only current document, select Current Document from the Look in drop down menu.

**Find Next:** Click this button to find the next match in the file.

**Replace:** Click this button to replace the next match in the file.

**Replace All:** Click this button to replace all matches in the file.

### Find Options

**Match case:** To limit your search to an exact case match, check the Match case checkbox. Otherwise, text will match regardless of case.

**Match whole word:** To limit your search to whole word matches, check the Match whole word checkbox. Otherwise, text will match in partial words.

**Search up:** To search from the current cursor position to the top of the file, check the Search up checkbox. Otherwise, the search will be from the current cursor position to the end of the file.

Use

This options allows you to search by regular expressions or by using a wildcard. See below for more information.

### Find using a Regular Expression

To search using regular expressions, check the **Use** check box and select **Regular Expressions**. You can use the following regular expression syntax:

|     |                                  |
|-----|----------------------------------|
| .   | Any single character             |
| *   | Zero or more                     |
| +   | One or more                      |
| ^   | Beginning of line                |
| \$  | End of line                      |
| \b  | Word boundary                    |
| \s  | White space                      |
| \n  | Line break                       |
| []  | Any one character in the set     |
| [^] | Any one character not in the set |

|   |                          |
|---|--------------------------|
|   | Or                       |
| \ | Escape special character |

### **Find using a Wildcard**

To search using wildcards, check the **Use** check box and select **Wildcards**. You can use the following wildcard syntax:

|     |                                  |
|-----|----------------------------------|
| *   | Zero or more of any character    |
| ?   | Any one character                |
| #   | Any single digit                 |
| []  | Any one character in the set     |
| [^] | Any one character not in the set |

### **Incremental Search**

Incremental search can be activated from the Advanced submenu of the Edit menu, or by pressing Ctrl+I (for forward incremental search) or Ctrl+Shift+I (for backward incremental search). The cursor icon changes to a binocular with an arrow indicating the search direction.

Begin typing the text that you want to search for. As you type, the editor highlights the first occurrence that matches the text. As you continue typing, the editor moves to the next match and highlights it. If no matches are available, the highlight will stop moving.

During incremental search, the following special keys are active:

| <b>Key</b>   | <b>Meaning</b>                                   |
|--------------|--|
| Esc          | Stop searching                                   |
| Backspace    | Remove the last character from the search string |
| Ctrl+Shift+I | Change the search direction                      |
| Ctrl+I       | Move to the next match                           |

### **Keyboard Shortcuts**

The Template Editor supports the following keyboard shortcuts:

| <b>Key</b>        | <b>Command</b>                       |
|-------------------|--------------------------------------|
| Ctrl+Enter        | Insert blank line above current line |
| Ctrl+Del          | Delete next word                     |
| Ctrl+Backspace    | Delete previous word                 |
| Ctrl+C, Ctrl+Ins  | Copy                                 |
| Ctrl+X, Shift+Del | Cut                                  |

|                      |                             |
|----------------------|-----------------------------|
| Ctrl+V, Shift+Ins    | Paste                       |
| Ctrl+Z               | Undo                        |
| Ctrl+Y, Ctrl+Shift+Z | Redo                        |
| Ctrl+Down            | Scroll down                 |
| Ctrl+Up              | Scroll up                   |
| Ctrl+Left            | Word left                   |
| Ctrl+Right           | Word right                  |
| Ctrl+PgUp            | Move to top of window       |
| Ctrl+PgDn            | Move to bottom of window    |
| Ctrl+}               | Move to matching bracket    |
| Tab                  | Indent line                 |
| Shift+Tab            | Outdent line                |
| Shift+Down           | Select down                 |
| Shift+Up             | Select up                   |
| Shift+Right          | Select right                |
| Shift+Left           | Select left                 |
| Ctrl+Shift+Right     | Select next word            |
| Ctrl+Shift+Left      | Select previous word        |
| Shift+Home           | Select to start of line     |
| Shift+End            | Select to end of line       |
| Ctrl+Shift+Home      | Select to start of document |
| Ctrl+Shift+End       | Select to end of document   |
| Shift+PgUp           | Select page up              |
| Shift+PgDn           | Select page down            |
| Ctrl+Shift+PgUp      | Select to top of window     |
| Ctrl+Shift+PgDn      | Select to bottom of window  |

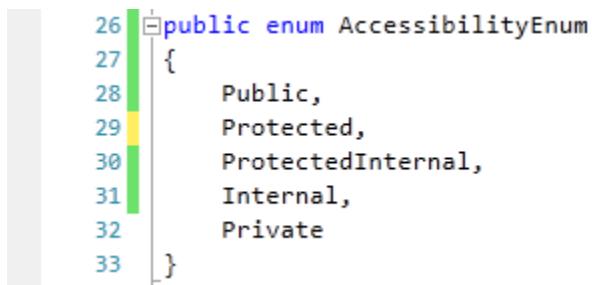


|              |                                |
|--------------|--------------------------------|
| Ctrl+A       | Select all                     |
| Ctrl+Shift+] | Select to matching bracket     |
| Insert       | Toggle overwrite mode          |
| Ctrl+O       | Open file                      |
| Ctrl+S       | Save file                      |
| Ctrl+Shift+S | Save all files                 |
| Ctrl+F4      | Close file                     |
| Ctrl+P       | Print                          |
| Ctrl+G       | Go to line                     |
| Ctrl+U       | Make lowercase                 |
| Ctrl+Shift+U | Make uppercase                 |
| Ctrl+I       | Incremental search             |
| Ctrl+Shift+I | Backward incremental search    |
| Ctrl+Shift+K | Toggle bookmark                |
| Ctrl+Shift+N | Next bookmark                  |
| Ctrl+Shift+P | Previous bookmark              |
| Ctrl+Shift+L | Clear bookmarks                |
| Ctrl+M       | Toggle outline expansion       |
| Ctrl+Shift+M | Toggle template code expansion |
| Ctrl+Shift+C | Insert code block              |
| Ctrl+Shift+V | Inverted code block            |
| Ctrl+Shift+W | Write block                    |
| Ctrl+Shift+Q | Script block                   |
| F7           | View code                      |
| F8           | View output                    |
| Ctrl+Shift+X | View Template Explorer         |

|              |                        |
|--------------|------------------------|
| Ctrl+Shift+D | View Schema Explorer   |
| F4           | View Properties Window |
| Ctrl+Shift+O | View Output Window     |
| Ctrl+Shift+R | Show Web browser       |
| Ctrl+Shift+B | Build                  |
| F5           | Run                    |
| F9           | Copy output            |
| F10          | Save output            |
| F1           | Help                   |

## Line Modification Markers

Lines of code that have been edited during the current session are indicated with a yellow line in the left margin of the editor:



```
26 public enum AccessibilityEnum
27 {
28     Public,
29     Protected,
30     ProtectedInternal,
31     Internal,
32     Private
33 }
```

When you save the file, the yellow markers turn green. Thus at any time, yellow markers show changed but unsaved lines of code, and green markers show changes in this session that have been saved.

## Outlining

Outlining provides a way to hide detail in your code until you want it.

Outlining in the Template Editor is of two types: automatic and manual. By default, Generator automatically creates outline blocks for every script block, property, enumeration, and object in your code. You can disable automatic outlining from the Outlining submenu of the Edit menu. You can also create manual outline blocks by inserting `#region` and `#endregion` (or in VB, `#region` and `#end region`) lines in your code.

To collapse an outline block, click the minus sign at the left margin of the block or place the cursor in the block and press `Ctrl+M`. To expand an outline block, click the plus sign at the left margin of the block or place the cursor in the block and press `Ctrl+M`.

In this code, Template Editor has automatically created outline blocks for the script block and the function, and the developer has added a manual outline with a region:

```
TemplateScript_cst | GetAccessModifier
3  <script runat="template">
4
5  #region My Custom Methods
6
7  public string GetAccessModifier(AccessibilityEnum accessibility)
8  {
9      switch(accessibility)
10     {
11         case accessibility.Public:
12             return "public";
13         case accessibility.Protected:
14             return "protected";
15         case accessibility.ProtectedInternal:
16             return "protected internal";
17         case accessibility.Internal:
18             return "internal";
19         case accessibility.Private:
20             return "private";
21     }
22 }
23
24 #endregion
25
26 public enum AccessibilityEnum
34
35 </script>
```

Clicking the minus sign to the left of the script block collapses the entire outline, and shows the script block grayed out. If you hover the mouse over the collapsed outline, Template Editor will display a tooltip with the first few lines of the code contained within the collapsed block:

```
2
3  <script runat="template"/>
```

Expanding the script block and then collapsing the region displays the text after the #region keyword:

```
3  <script runat="template">
4
5  My Custom Methods
25
26  public enum AccessibilityEnum
34
35  </script>
```

Expanding the region and collapsing the function displays the function name.

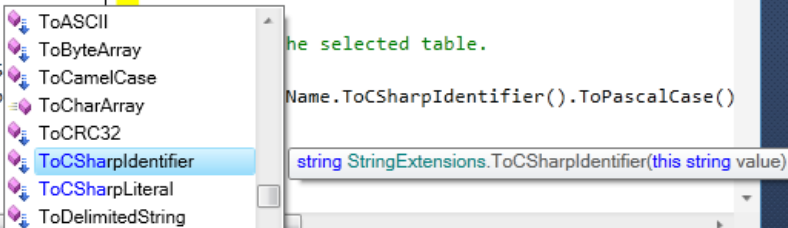
```
3  <script runat="template">
4
5  #region My Custom Methods
6
7  public string GetAccessModifier(AccessibilityEnum accessibility)
23
24  #endregion
25
26  public enum AccessibilityEnum
34
35  </script>
```

## Statement Completion

Statement completion is very similar to the IntelliSense feature of Visual Studio. With Statement Completion, Template Editor prompts you with identifier names as you type.

You can bring up the completion lists at any time within a code block by pressing 'Ctrl+Space'.

```
<%@ Import Namespace="CodeSmith.Core.Extensions" %>
public class SourceTable {
    // Generate a property for
    <% foreach (var column in SourceTable.Columns) %>
    public string ColumnName {
        get {
            return column.SystemType
        }
    }
}
```



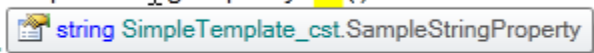
The dropdown list of identifiers appear as soon as you type the dot after an object name. To select an identifier, click it with the mouse or select it with the arrow keys or by typing enough letters to uniquely identify it and then type a space or other separator character to auto-complete.

Statement completion depends on reflection to gather the information that it presents. This means that your template must compile successfully for statement completion to work on types defined in the template. Types defined in an external assembly will show up in the statement completion lists whether the current template compiles or not.

### Quick Info

Automated quick info tips show whenever the mouse is hovered over words such as identifiers. The quick info tips display detailed information about the related type, member, variable, etc. Similar to parameter info tips, all information is presented using rich text formatting.

```
private void SampleStringProperty()
{
    // Do something
}
```



### Parameter Info

Automated parameter info tips show whenever typing an invocation (E.G., a method call). The tips show detailed information about the invoked member along with details about the current argument being typed. In the case where the invoked member has multiple overloads, arrows show on the popup and allow toggling between all the available overloads.

```

3 <%= GetRandomNumber(4); %>
4
5 <script>
6     int ParameterInfo_cst.GetRandomNumber(int seed)
7     A reimplimentation of Sony's random number generator.
8     seed: A random seed.
9     </script>
10    <summary>
11    </summary>
12    <param name="seed">A random seed.</param>
13    <returns>A random number based off the seed.</returns>
14    public int GetRandomNumber(int seed)
15    {
16        return 4;
17    }
18 </script>

```

As with quick info, parameter info can handle rich-formatted content display using HTML-like markup tags. Colors and font weights or styles can be used to bring attention to portions of the info tip. The screenshot above shows the font weights bringing attention out to the current parameter **seed**.

Multiple signature options can be displayed in a single parameter info tip. In those scenarios, arrows automatically appear that can be clicked. Alternatively the end user can use the up/down arrow keys to switch between options.

### Anonymous Types

In previous versions of CodeSmith Generator there was no support for anonymous types or extension methods. The sample below will show off an example of Anonymous Type support.

```

7 var numbers = new System.Collections.Generic.List<int>();
8
9 var anonymousType = new {
10     Numbers = numbers,
11     Name = "Blake"
12 };
13
14 string name = anonymousType
15 }
16

```

In this screenshot, we have implicitly declared a variable **anonymousType** as an anonymous type. The anonymous type's properties are initialized using a variable (**numbers**) as well as a string constant ("Blake"). Hovering over the **anonymousType** variable will display quick info, which reveals **anonymousType** as an anonymous type.

```

7 var numbers = new System.Collections.Generic.List<int>();
8
9 var anonymousType = new {
10     Numbers = numbers,
11     Name = "Blake"
12 };
13
14 string name = anonymousType.
15 }
16 </script>

```

- Equals
- GetHashCode
- GetType
- Name**
- Numbers
- Tostring

string <anonymous type>.Name

We can bring up the statement completion list by typing period after our variable. This will display a list that contains all of the members of the variable **anonymousType**. The two properties we declared (Numbers and Name) in the anonymous type's creation expression appear in the list. You can see it has correctly assigned property names and their types.

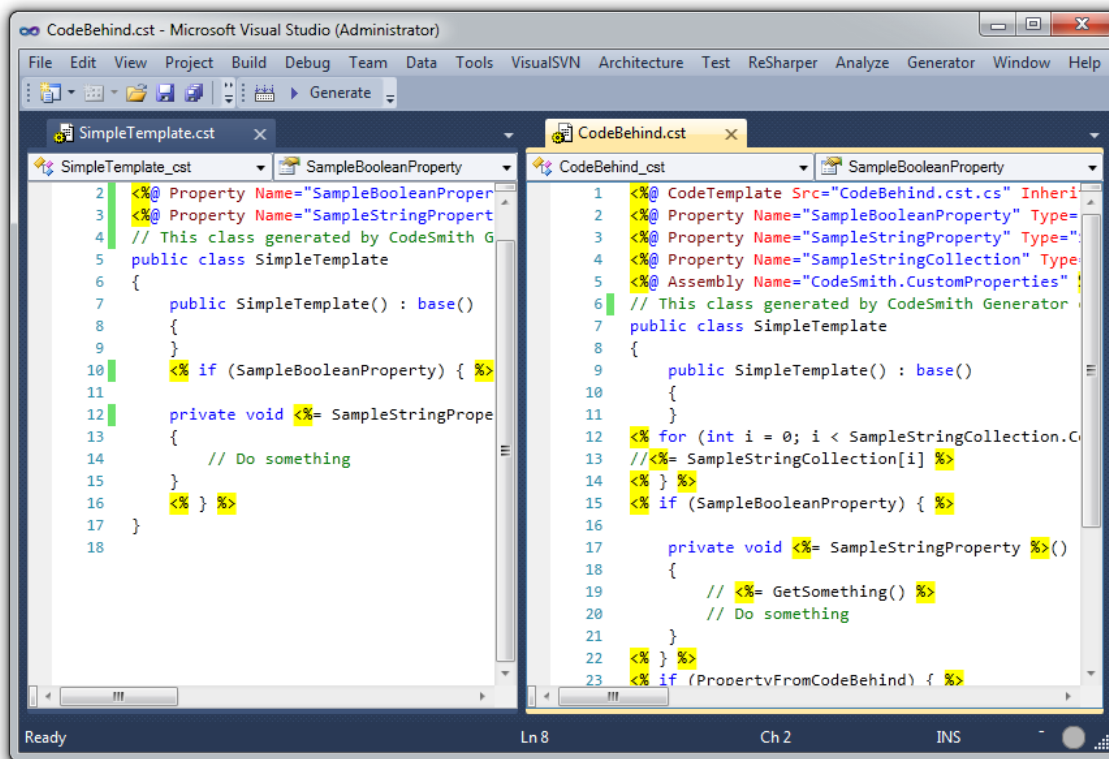
## Tab Groups and Split Windows

The Template Editor provides you with two ways to edit in multiple locations at the same time. Tab groups allow you to open two or more files for editing. Split windows allow you to have two editing panes open in to the same file at the same time.

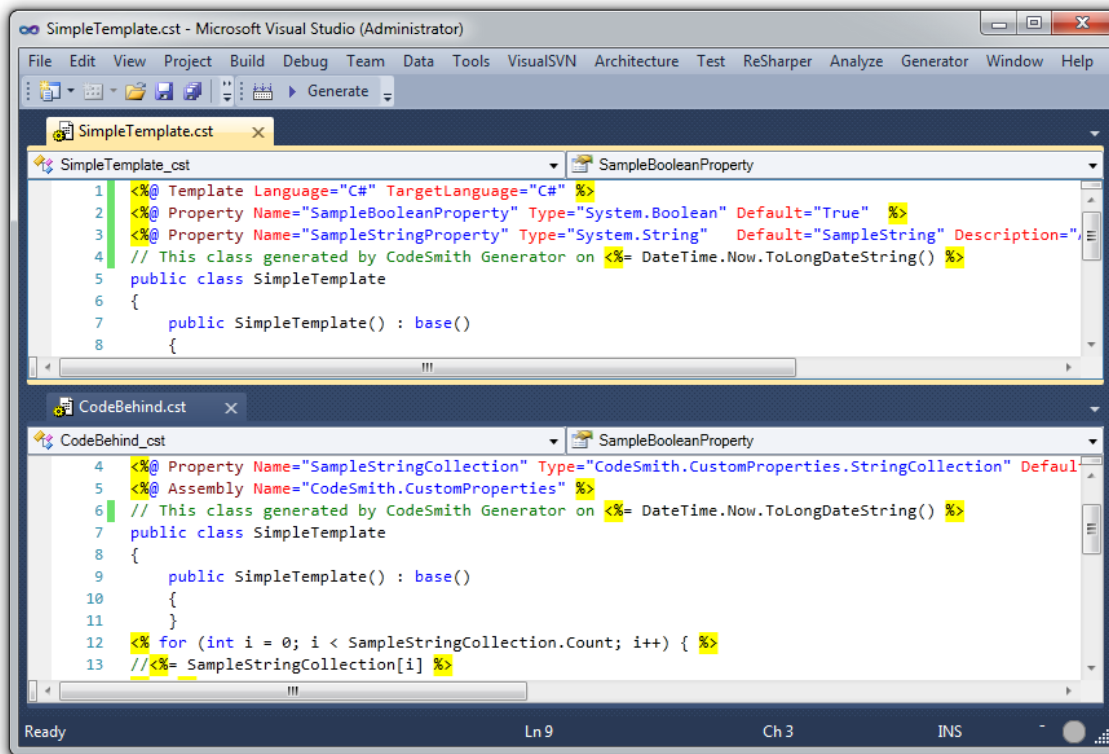
### Tab Groups

To create a tab group, select New Horizontal Tab Group or New Vertical Tab Group from the Window menu. Alternatively, click and hold the mouse cursor on an existing tab at the top of an open document window, and drag the tab down into the code-editing area. When you drop the tab, the shortcut menu will offer you the choice of creating a new vertical tab group or a new horizontal tab group. In any case, the active tab when you perform the operation will become the first tab in the new tab group.

#### Vertical Tab Groups



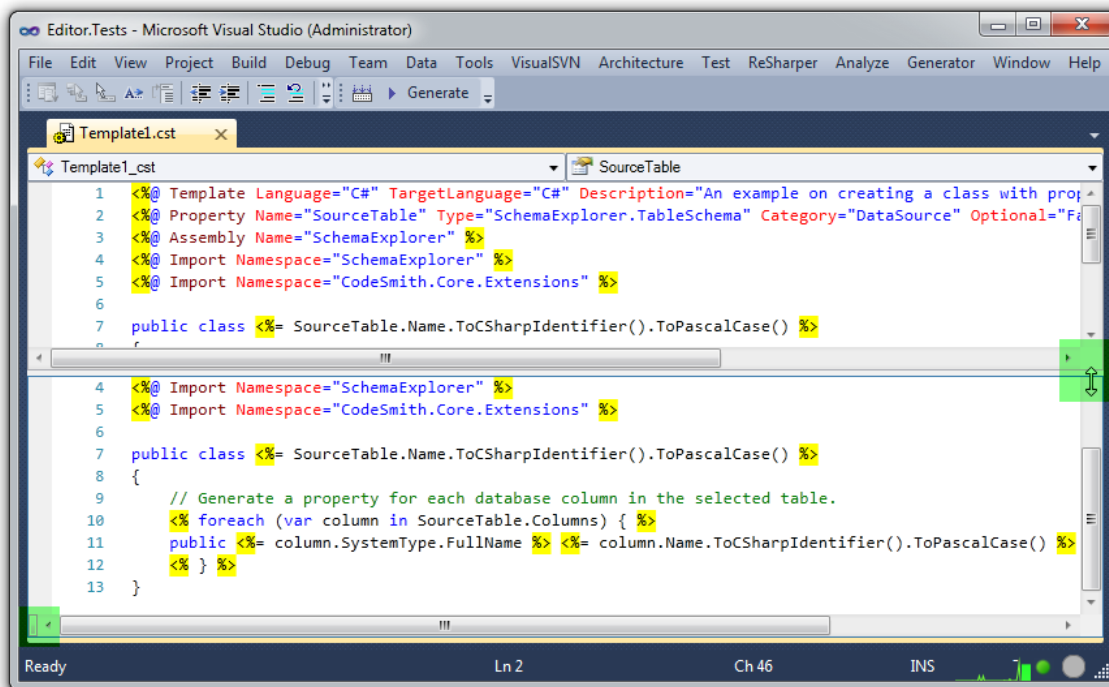
#### Horizontal Tab Groups



You can move a tab from one tab group to another tab group by dragging and dropping the tab. You can also create additional tab groups by repeating the process that creating the initial tab group. However, all tab groups must be of the same type; you cannot mix horizontal and vertical tab groups. If you close or remove the last tab in a tab group, Template Editor will eliminate the tab group.

### Split Windows

To create a split window grab the splitter handle at the top of the vertical or horizontal scroll bar (highlighted in green in the image below) with the mouse and drag it downwards. The result will be a single window with two panes, both of which open on the same file:



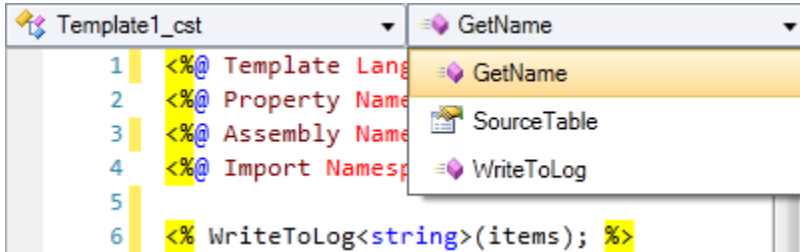
You can scroll and edit in the two panes of a split window independently. Because the two panes are both views into the same underlying file, any changes in one pane are immediately reflected in the other pane.

## Template Navigation

The Go To Definition, Navigation bar, View Code features makes navigating around your template quick and easy.

### Navigation bar

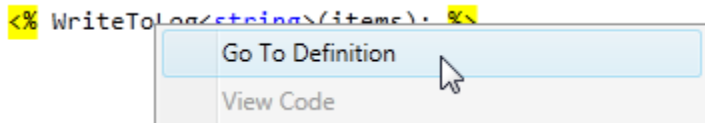
The Navigation bar is located at the top of the [template document](#) and displays the types and members in the current template. Types are always shown in the left drop down menu and members are always shown in the right drop down menu.



When you select a type from the drop down, the caret is placed on the first line of the type. The same is also true when you select a member. The drop down boxes are immediately updated to reflect the current location of the caret.

### Go To Definition

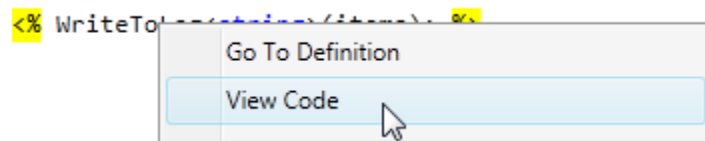
This feature will allow you to right click on any identifier (E.G., Classes, Methods, Properties and Variables) in your template and quickly navigate to where they were defined. This eliminates the need to use Find or scrolling around your template to navigate to find a property or method you were looking for.



To use Go To Definition, just right click on an identifier or press F12.

### View Code

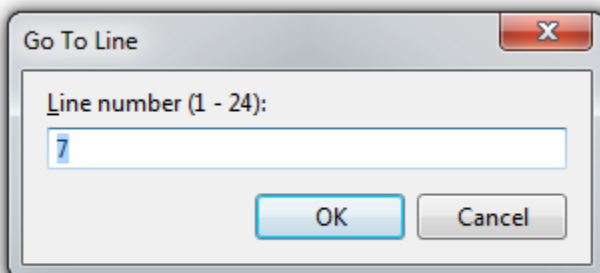
This feature allows you to open a code behind or partial class that is defined via a [CodeTemplates Src attribute](#). You can quickly navigate to the code behind or partial class without opening up [Template Explorer](#) or cluttering the [Template Editor](#).



To use View Code, just right click anywhere in the [template document](#) or press F7.

### Go To Line

This feature will allow you to jump to any line in the [template document](#). This is very useful when you are debugging an error and you know the exact line number you wish to navigate to.



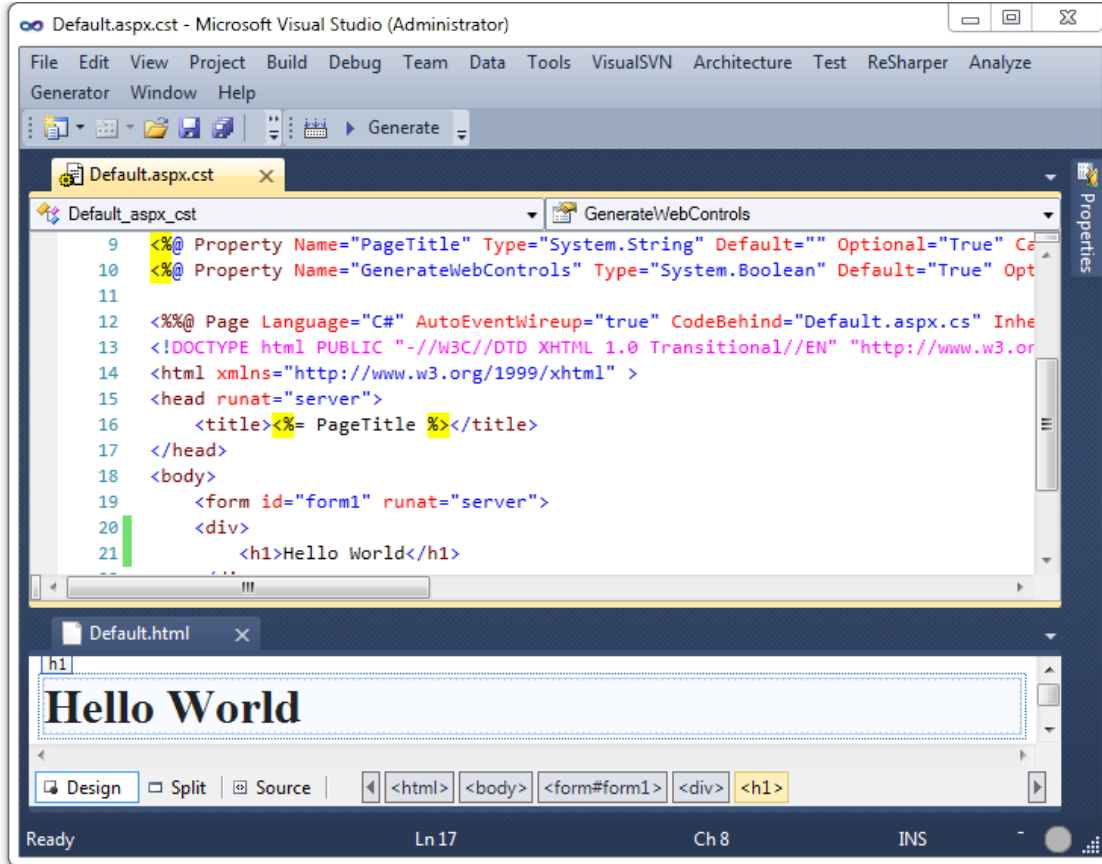


To use Go To Line, select Go To from the Edit menu or press Ctrl+G.

## Themes and Syntax Highlighting

### Syntax Highlighting

The Template Editor support syntax highlighting for many different language types like C, C++, CSharp, CSS, Visual Basic, Html, Java, JavaScript, Perl, SQL, Xml and many more.



You can see the template document and generated document in the above screenshot. As you can see the generated document (Default.html) is being shown in the html designer. This allows you to see the generated documents design view right after you generate!

The Template Editor also supports copying your Syntax Highlighting color schemes to RTF and HTML.

```
<%%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApplication._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

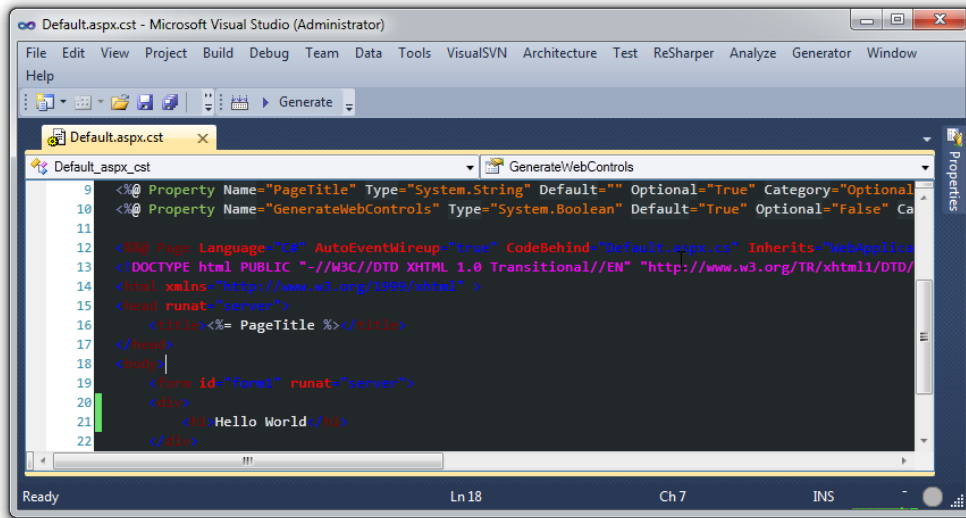
<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">
```

When you copy any part of a template document to the Windows Clipboard, an HTML and RTF copy is also placed in the clipboard. This means that when you paste into an Rich Text box, your formatting will also be copied as shown above.

### Themes

The Template Editor also supports themes. Template Editor uses the configuration settings to determine what colors to use.



## Building, Running, and Compiling Templates

CodeSmith Generator allows you to build, run, and compile your templates.

### Building Templates

To compile the current template, select Build from the Build menu, or press F6, or press Ctrl+Shift+B, or click the Build button on the toolbar. The Output window will show the results of the build operation.

### Running Templates

To run the current template, select Run from the Debug menu, or press F5, or click the Run button on the toolbar. This will execute the current template, using values from the Properties window together with the template to create the generated code. At the end of the Run operation, CodeSmith Generator will display the generated code in a generated document.

### Compiling Templates

You can compile a template to an assembly (.dll file), which you can then reference from any .NET project. To compile a template to an assembly follow the steps below.

1. Create or use an existing .NET 4.0 class library project.
2. Add your master template to the project via the add new or add existing. You can also add the template as a linked item.
3. Ensure the ClassName and Namespace attributes on the CodeTemplate Directive are set properly on your template.
4. Right click the template you added to the project and select properties.
5. Once the properties window is open, set the templates CustomTool to TemplateSourceGenerator.
6. Build the project you created in step 1

## Customizing CodeSmith Generator

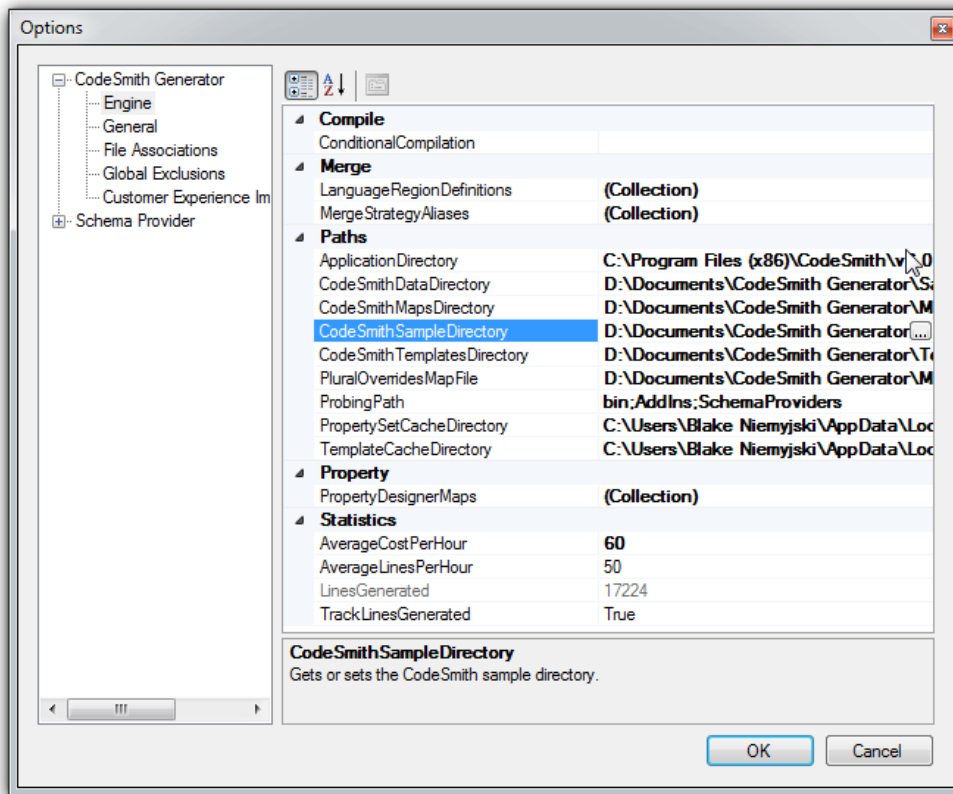
There are several reasons you might want to customize CodeSmith Generator. You might want to reset file associations or change the Sample Directory location. You can configure all of CodeSmith Generators settings through the Options dialog.

### Opening the Options dialog.

You can open the Options dialog from the Generator menu. You can also open up the Options dialog by launching the csconfig.exe executable located in the CodeSmith Generator Program Files folder (C:\Program Files\CodeSmith\<VERSION>).

### Configuring Settings

All of the configuration settings are grouped into sections and are displayed in nodes on the left hand side as shown in the image below.



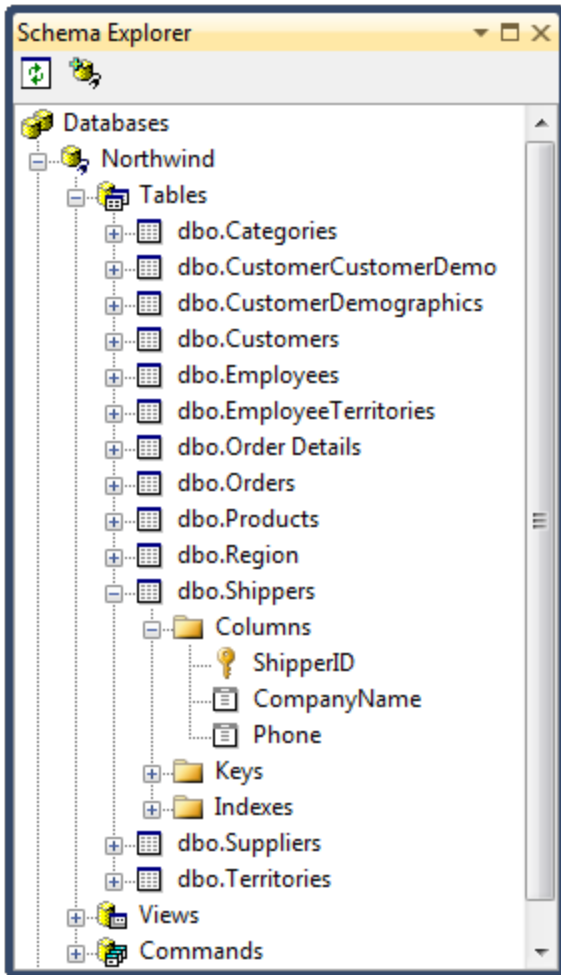
In the above image, the Engine node is selected and allows you to configure various settings like the location of the Sample Directory (CodeSmithSampleDirectory). To close the Options dialog, press OK to apply the changes or Cancel to ignore the changes.

**i** You may need to restart CodeSmith Generator after changing property values for the changes to take effect.

**o** Some properties values may be persisted in real time and will not be rolled back when Cancel is clicked.

## Using Schema Explorer

Schema Explorer provides an easy and graphical way to explore the schema of databases. This gives you the ability to manage [extended properties](#) or retrieve the names of tables, views, commands, and their components.



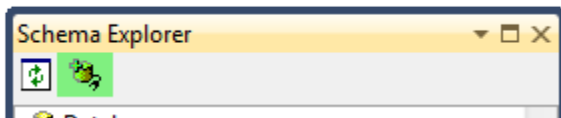
## Opening Schema Explorer

You can open Schema Explorer by selecting **Schema Explorer** from the Generator menu.

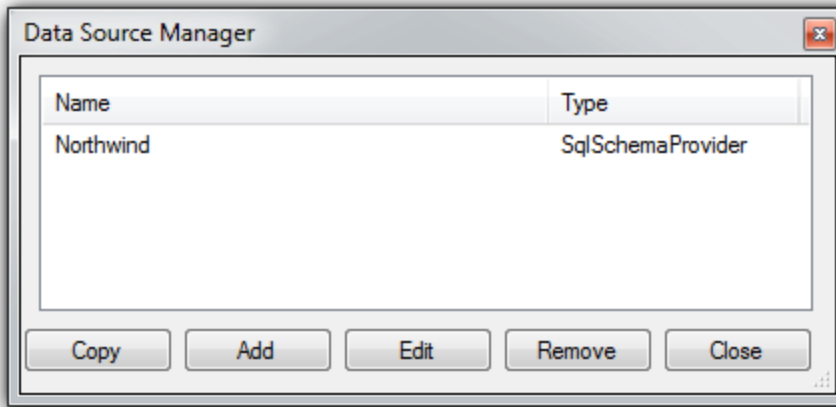
## Adding a new database connection


The following steps will show you how to add a new database to Schema Explorer.

The first step is to click the Manage Data Sources button as highlighted in green.

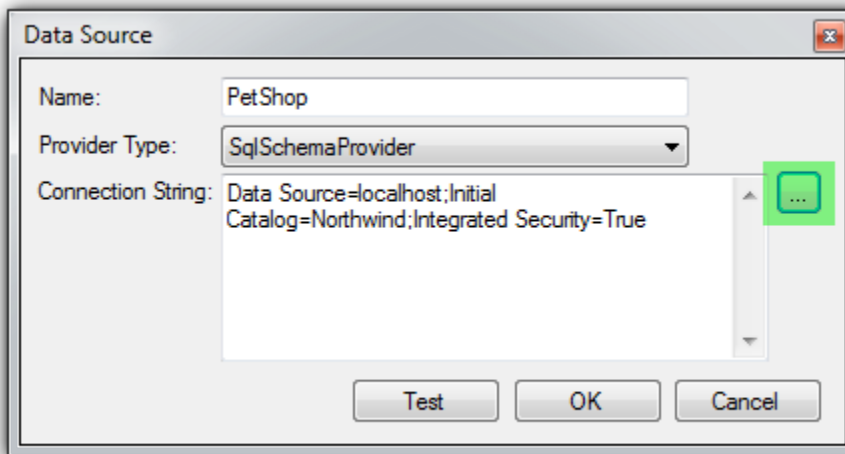



This will open the Data Source Manager dialog box.




 The Data Source Manager allows you to Add/Edit or Remove existing Databases.

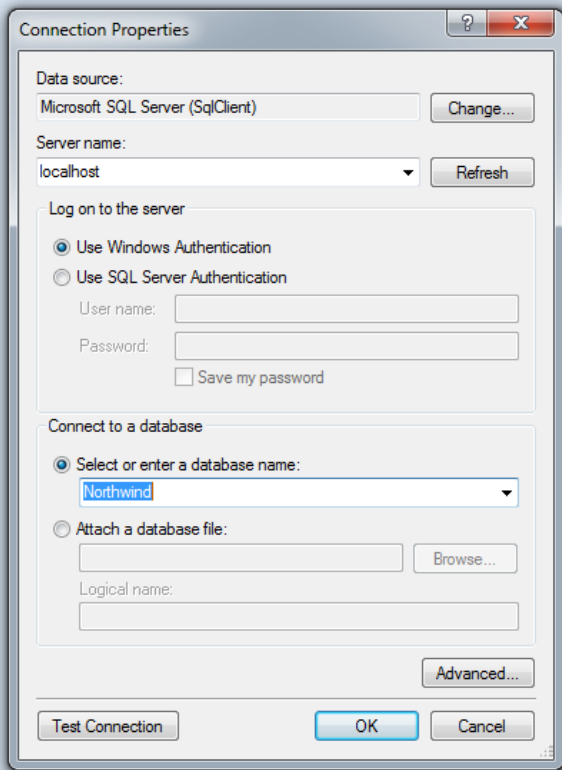
Next, click the Add button to open the Data Source dialog box.



 The Test button allows you to test the defined Connection String.

Next, enter a name for the new data source, select an appropriate provider, and enter a connection string.

 If a connection string designer is available, the ellipsis button (...) highlighted above in green will be enabled.




The above image shows the SQL Server Connection String Editor.

Finally, you can click the OK button to create the newly defined database and click the close button to close the Data Source Manager.

## Managing Extended Properties

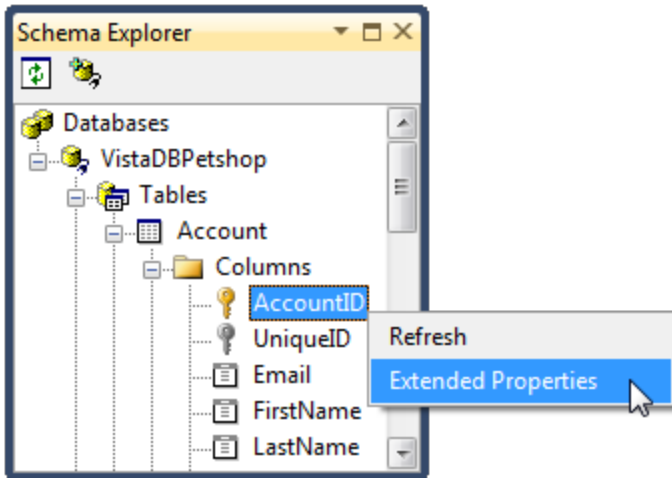
Managing your RDBMS' Extended Properties has never been easier. You can now Add/Edit, or Manage a new schema extended property directly inside of Schema Explorer.

This is a very powerful way to add custom meta-data to the already feature rich meta-data that Schema Explorer provides. You can now easily add database object descriptions, or anything that will help you drive your generation process using a powerful storage on the database server.

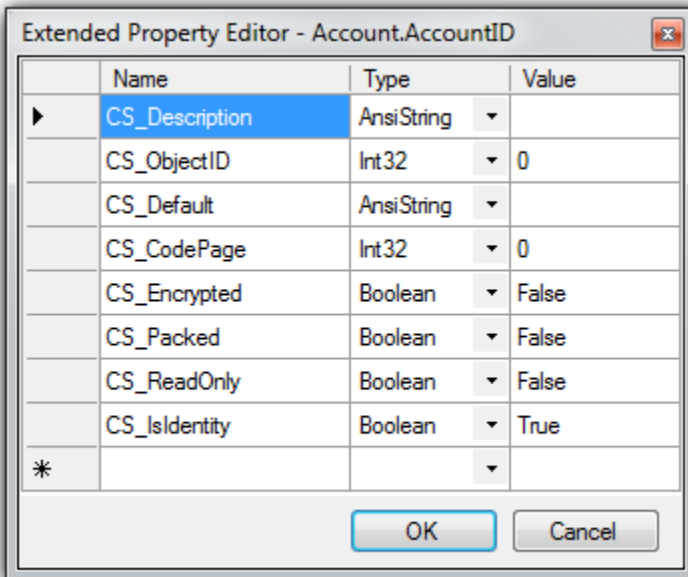
 [Click here](#) for more information on using Extended Properties in Templates.

## Managing Extended Properties

To manage your Extended Properties, simply open up the Schema Explorer window.



Once the Schema Explorer window is open, continue to expand nodes until you find the Schema Object you are looking for. In the image above selected the AccountID Column for editing. To bring up the **Extended Property Editor**, just Right-Click on a Schema Object and select **Extended Properties**.



Items beginning with "CS\_" and "MS\_" are read-only and are used for CodeSmith's Generator meta-data.

### Removing an Extended Property value

You can remove any Extended Property by right clicking a row and selecting delete.

### Editing an Extended Property value

You can edit an Extended Property value that is not marked as a read-only by editing the rows Value field.

### Removing an Extended Property value

To add a new row just start typing below the last row of data in the Name column.

### Saving Extended Property Values

To save your Extended Property values, just click on the **OK** button. To ignore your changes click on the Cancel button.



Some Schema Providers may not implement this feature and throw a Not Supported Exception when retrieving or saving Extended Property values.

## Using the Map Editor

CodeSmith Generator Maps allow developers to reduce the amount of plumbing code in their templates and increase the readability and reusability at the same time. CodeSmith Generator Maps also provide an easy way to manage dictionary maps for doing word translation lookups in code. This used to be a frequent and cumbersome challenge that a template writer must face when trying to map types from disparate systems.

### Map Editor

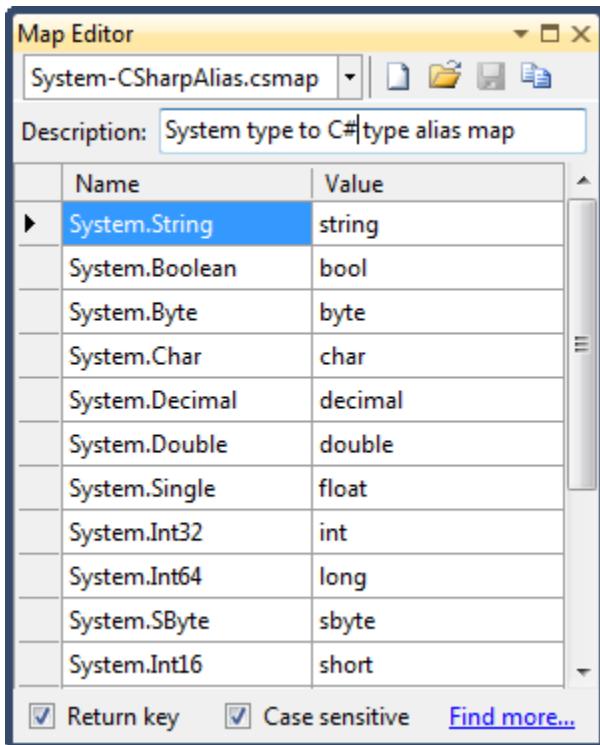
Generator includes a Map Editor that makes it very easy to create and manage your own CodeSmith Generator Map lists. Generator also ships several of the most common mapping scenarios and it's output is in a familiar XML format allowing the developer community to contribute and share maps they create as well through a new [map file gallery](#) at the [CodeSmith community](#).

### Opening the Map Editor

You can open the Map Editor from the Generator menu.

### User interface elements

The section below will walk you through the various user interface elements of the Map Editor.



(New File)

**Name:** The map name is also the file name so consider choosing a name that is fairly descriptive of what the map's intent is. This drop down allows you to edit and view previously defined map files.



If you select (New File) then a new mapping file will be created.



When Generator attempts to discover a map by name, it first looks in the configuration directories, the template directories and you can also give it a relative or full path.





**New File:** Clicking this button allows you to create a new mapping file.



**Open File:** Clicking this button will open a Windows Explorer Browse dialog and let you pick a mapping file that exists anywhere on your hard drive.



**Save Copy:** Creates a duplicate map file of the currently selected map file.

**Description:** A description field of what the maps intent is for.

**Return Key:** The return key check box indicates whether or not to return the original key from the map if the key is not found within the map. This is important because there can be many situations where there might not be an entry for the key and it's value, but you would still like the key returned instead of null.



In the image to the above, it's converting a fully qualified type to it's C# equivalent keyword. In cases where one doesn't exist, such as System.DateTime or System.Guid, you could still use the original key, System.DateTime, which would be completely valid.

**Case Sensitive:** The Case sensitive check box determines whether or not to search for the key and consider case sensitivity. Often, word dictionaries do not require case sensitivity since most use keys that are unique by name.

**Find More:** Launches a browser session to browse the online gallery of community collaborated maps.



Click [here](#) to learn more about developing using a CodeSmith Generator Map.

## Developing using a Generator Map

While developing templates, a common scenario developers face is accessing a lookup list based on some sort of information. This document will cover the definition of a Map Directive and then will walk you through an example of using a Map file.

### The Map Directive

Using a Map directive is quite easy and very flexible, all you need to do is define a Map Directive. For example, this map directive defines a template property of type MapCollection that is named CSharpAlias. This CSharpAlias property will be populated with the mapping file values declared in the System-CSharpAlias.csmmap.

```
<%@ Map Name="CSharpAlias" Src="System-CSharpAlias" Description="System to C# Type Map" %>
```

The System-CSharpAlias.csmmap is resolved by looking in the current template directory as well as Generator Maps folder (Documents\CodeSmith Generator\Maps)



Mapping files are resolved by looking in the current template directory as well as Generator Maps folder (Documents\CodeSmith Generator\Maps).

### Map Directive Attributes

The Map directive has five possible attributes. The Name and Src attributes are required, and the other attributes are optional.

#### Name

The Name attribute references name of the map specified to use in code.

#### Src

The Src attribute defines the file name of the map file or the file path to the map file.



Adding the extension name is not required.

### **Description**

The Description attribute supplies descriptive text to be displayed at the bottom of the property sheet when this property is selected.

### **Reverse**

When you require to translate the lookup back to the key from the value, you can reverse the map. You simply load the collection using the reverse overloaded method.

### **Default**

The Default attribute defines the default action to take place when a key is not found. If set to True, a default value will be returned when the key is not found.

### **API Usage**

You can also use CodeSmith Generator Maps without using a Map Directive. This means that you are creating and populating a new MapCollection instance through code. You can interface with a map from code simply by loading the map by name.



Mapping files are resolved by looking in the current template directory as well as Generator Maps folder (Documents\CodeSmith Generator\Maps).

### **Common API Usage**

This mimics the usage of the mapping used in the example below which uses a declarative model.

```
MapCollection list = MapCollection.Load("System-CSharpAlias.csmap");  
list.ReturnKeyWhenNotFound = true;  
Response.Write(list[column.SystemType.FullName]);
```

### **Reverse Map**

Call the overloaded Load method when requiring to load the map with the key value pairs swapped.

```
MapCollection list = MapCollection.Load(string mapName, bool reverseMap);  
Debug.Assert(list[myValue] == myKey);
```



See the CodeSmith Generator API help for more API Coverage.

### **Example**

A common example is a mapping between CLR data types and SQL Server data types. Before CodeSmith Generator Maps, this functionality would have been accomplished by writing a method with a long switch/Select Case statement as shown in the example below.

```
public string GetFrameworkType(DataObjectBase column)
{
    switch(column.DataType)
    {
        case DbType.AnsiString:
        case DbType.AnsiStringFixedLength:
        case DbType.String:
        case DbType.StringFixedLength:
            return "String";

        case DbType.Binary:
            return "Byte[]";

        case DbType.Boolean:
            return "Boolean";

        case DbType.Byte:
            return "Byte";

        case DbType.Currency:
        case DbType.Decimal:
        case DbType.VarNumeric:
            return "Decimal";
    }
}
```

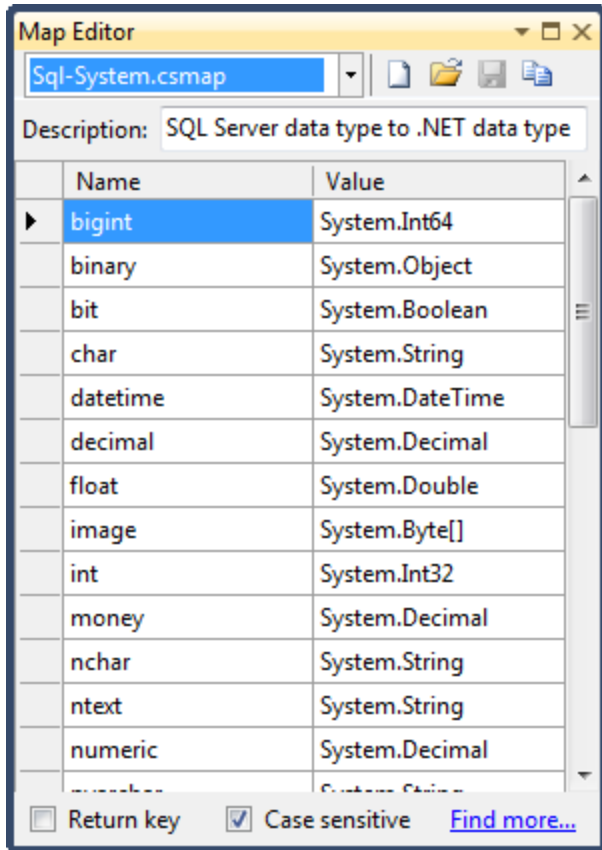
Using CodeSmith Generator maps reduces the amount of code needed for lookups and also allows you to share mapping logic between templates. We can convert this code into a mapping file by creating a new map file.

### ***Creating the map file***

The first step is to [open the Map Editor](#) and create a new map file by [clicking on the New File button](#).

### ***Populating the map file***

Next, we will start by filling out the name and description of our map file along with entering in the values defined above. The end result should look something like this:



**i** This mapping file ships with CodeSmith Generator.

### Updating your template to use a map

We can start by removing the switch/Select Case statement defined above. Doing so will cause build errors (after building the template) any place where this method was being called. This allows us to use the Error Window to navigate to the usages of the previous code and update the usages to use a mapping file.

### Adding a Map Directive

Using a map is quite easy and very flexible. You simply register the map using the Map Directive (also highlighted in green in the image below).

```
<%@ Map Name="CSharpAlias" Src="System-CSharpAlias" Description="System to C# Type Map" %>
```

### Updating usages

Once the map has been registered, you simply reference the map and pass it the value similar to using another Collection class.

As you can see on Line 13 we use the mapping file by using the map name (E.G., **CSharpAlias**) as defined in the Map Directive on line 7. Previously the code on Line 13 would have called the GetFrameworkType(column) method.

```

1  <!--
2  Name: Database Table Properties
3  Author: Paul Welter
4  Description: Create a list of properties from a database table
5  -->
6  <@ CodeTemplate Language="C#" TargetLanguage="C#" Debug="False" Description="Create a list of
7  <@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" Category="Context" Descript
8  <@ Map Name="CSharpAlias" Src="System-CSharpAlias" Description="System to C# Type Map" %>
9  <@ Assembly Name="SchemaExplorer" %>
10 <@ Import Namespace="SchemaExplorer" %>
11
12 <% foreach (ColumnSchema column in this.SourceTable.Columns) { %>
13 private <%= CSharpAlias[column.SystemType.FullName] %> _<%= StringUtil.ToCamelCase(column.Nam
14
15 public <%= CSharpAlias[column.SystemType.FullName] %> <%= StringUtil.ToPascalCase(column.Name
16 {
17     get { return <%= StringUtil.ToCamelCase(column.Name) %>; }
18     set { _<%= StringUtil.ToCamelCase(column.Name) %> = value; }
19 }
20
21 <% } %>

```


Mapping is Great!  
 Highlighted Green shows the usage of Mapping.

## Using CodeSmith Generator Projects

CodeSmith Generator Projects manage groups of CodeSmith Generator templates and their outputs all in a single CodeSmith Generator Project file (.csp). CodeSmith Generator Projects are files that enable you to run an entire generation process at anytime in a simplistic manner from many different environments.

### About

A CodeSmith Generator Project file uses a .csp file windows extension, and stores XML metadata about your CodeSmith Generator Project.

 Learn more by reading [Anatomy of a Project File](#).

A CodeSmith Generator Project file can be generated or configured by right clicking on on a CodeSmith Generator Project File (.csp).

### Generation Capabilities

CodeSmith Generator Project files enable the management and execution of a generation process in many environments.

### Windows Explorer and Template Explorer

Managing a CodeSmith Generator Project right from Windows Explorer is simple and doesn't require you to use CodeSmith Generator to manage a project. Options are available through the right-click context menu in your CodeSmith Generator Project file (.csp). The menu options include:

1. [Manage Outputs](#) - Gives the ability to manage your CodeSmith Generator Project.
2. [Generate Outputs](#) - Will kick off the generation process to produce outputs configured in your CodeSmith Generator Project.
3. [Add Outputs](#)

 Learn more by reading [Using CodeSmith Generator Project from Windows Explorer](#).

### Command-Line


You can Generate Outputs of a CodeSmith Generator Project in the command line by using the [using the CodeSmith Generator Console Application](#). You would simply call:

```
cs MyCodeSmithProject.csp
```

## Visual Studio

The tight integration with Visual Studio allows you to fully manage any CodeSmith Generator Project right from Visual Studio! This means you can maintain a high Code Generation presence right within Visual Studio and not have to switch applications to run code generation.

1. [Manage Outputs](#) - Gives the ability to manage your CodeSmith Generator Project.
2. [Generate Outputs](#) - Will kick off the generation process to produce outputs configured in your CodeSmith Generator Project.
3. [Add Outputs](#)
4. [Output Options](#) - Only available in Visual Studio, and allows you to control the output options after generation.

 Learn more by reading [using a CodeSmith Generator Project inside Visual Studio](#).

## MSBuild

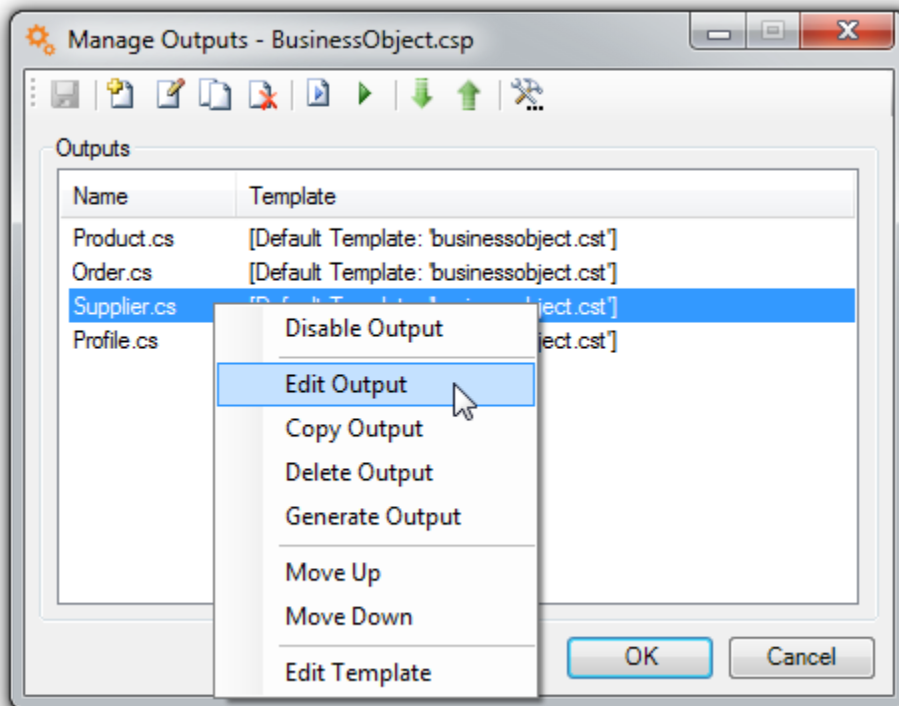
You can create your own custom pre-generation build logic by utilizing the CodeSmith Generator Task within MSBuild. MSBuild tasks help manage the build process within your Visual Studio projects.


There might be times when you need to customize some aspect of the generation process during it's consuming build process. During these time you might have to call CodeSmith Generator from MSBuild using the CodeSmith Generator task that's shipped for you.

 Learn more by reading [using a CodeSmith Generator Project from MSBuild](#).

## Manage Outputs

Managing your CodeSmith Generator Project is simple to do, and best of all, you don't have to be in CodeSmith Generator to do it. That's why we've exposed the ability to manage your Generator Project File from just about anywhere, Windows Explorer, and Visual Studio. Meaning whatever interface you're seeing the project file, you can Add, Edit or Delete via the Manage Outputs dialog. Code Generation has never been easier.



 You can right click on an output to bring up the menu shown above or double-click to edit.

In the following sections we will show you how to use the full power of Manage Outputs.

## Configuring your Options

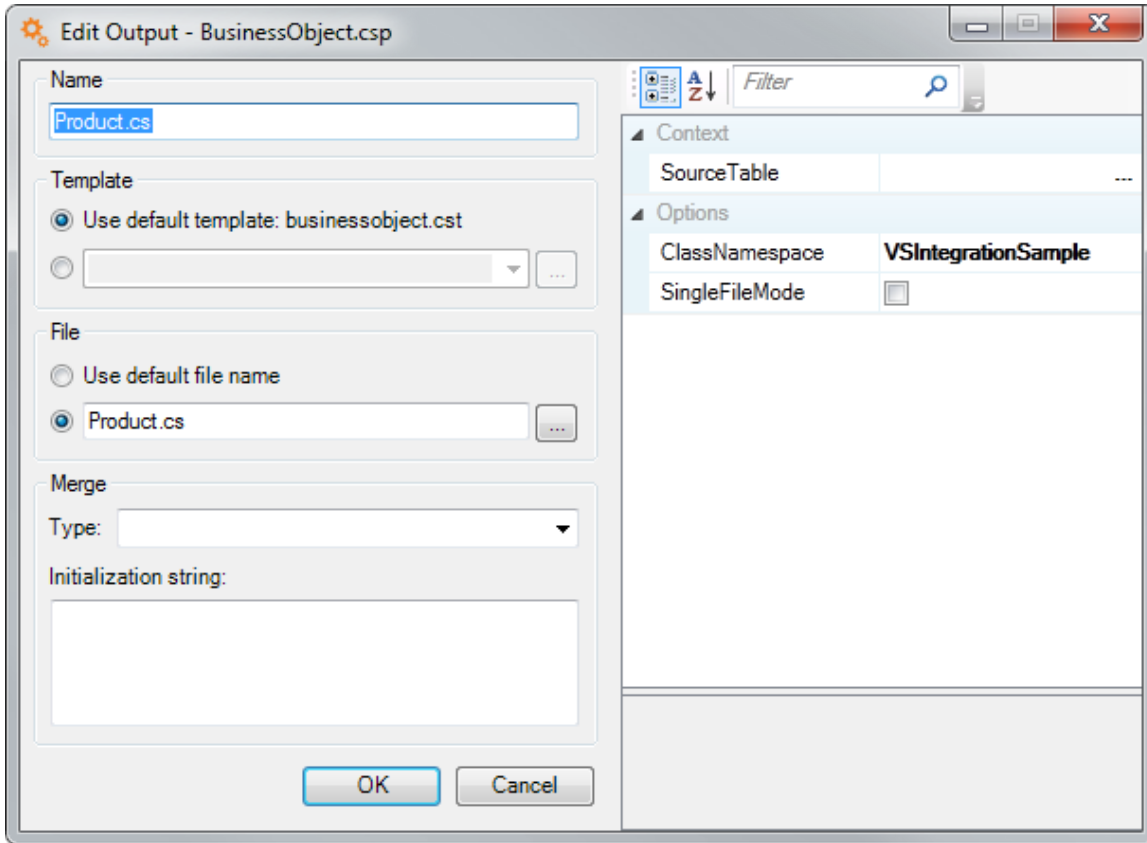
There are several options you have the ability to configure, initially, you need to add at least one Output.

|                                 |   |
|---------------------------------|---|
| <b>Outputs Window</b>           | Are the current configured Outputs for your CodeSmith Generator Project. If an output does not specify a CodeSmith Generator template to use, the Output will use the Default Template. (depicted above)  |
| <b>Add Button</b>               | Opens the Add Output Form so that you can add a new output with an existing template or use the default template. For more information read the Add/Edit Outputs section below.   |
| <b>Edit Button</b>              | Opens the Edit Output Form which you can edit the output options or template properties. For more information read the Add/Edit Outputs section below.  |
| <b>Copy Button</b>              | Creates a copy of a selected output. This is especially useful if you are creating many outputs that use the same template, but use a single different piece of meta-data to differentiate from. You would then only have to change the one property for all of the copied outputs. |
| <b>Delete Button</b>            | Deletes an Output from your CodeSmith Generator Project.  |
| <b>Generate Selected Button</b> | Generates the output for the selected template.   |
| <b>Generate Button</b>          | Begins the generation process for this CodeSmith Generator Project. This is the same as selecting Generate Output.  |
| <b>Project Options Button</b>   | Opens the <a href="#">Project Options</a> dialog.   |
| <b>OK Button</b>                | Persists all modifications that were made on the form.  |
| <b>Cancel Button</b>            | Cancel any changes from the form and closes the window.   |

## Add/Edit Outputs

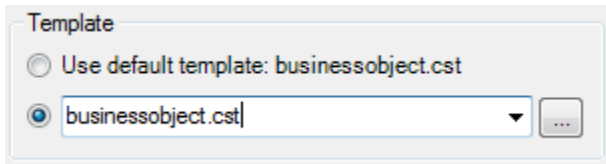
To open the Edit Output dialog, you can click on the edit icon in the Manage Outputs toolbar or double click on an output listed in the Outputs list.

There are four panels that cover all of the options for configuring an Output. Below is the entire Add/Edit Output Form.



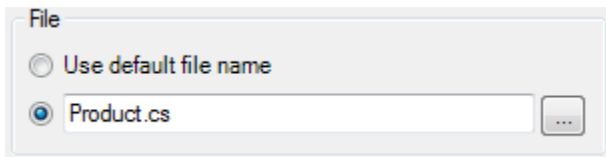
### Choose a Template


If a Template has been configured from in the Main Manage Outputs screen, then you will see the template name listed under the Use Default Template option. Otherwise, you will have a File Chooser control to select your template that you want the output to be generated from.



### Choose an Output

Choosing an output requires you to name the Output file you are wanting to use. The default Output name is created by the template you are using.

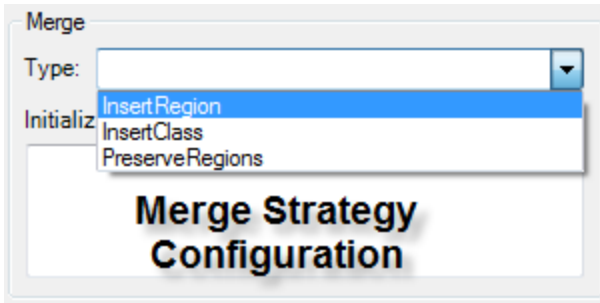


 If you have a template that does not have an output, this field will be ignored.

### Optional: Choose a Merge Strategy

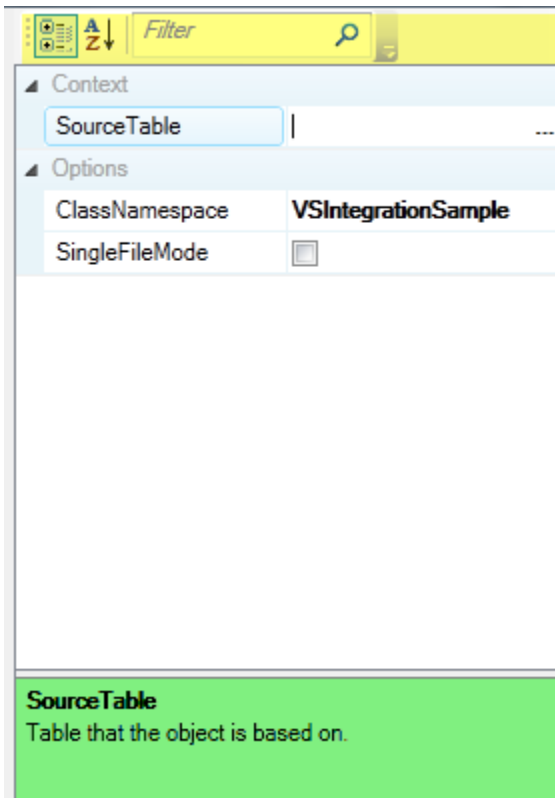
If your template uses a [merge strategy](#), either InsertRegion or PreserveRegions, you would enter the pattern here.





### Configuring the Property Sheet

The right pane is a familiar property of the template you have chosen. You would fill in the necessary values for your template to run. The top shaded yellow area shows the [property sheet options](#).



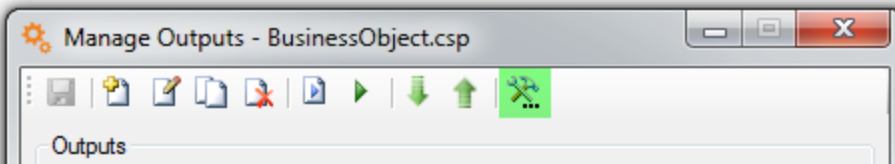
Any Required fields missing will throw an error when you attempt to save the Output changes.

### Project Options

The [Project Options](#) dialog allows you to configure behavior for the project. Some of the settings include a default template, single or multiple file output, default properties and project variables.

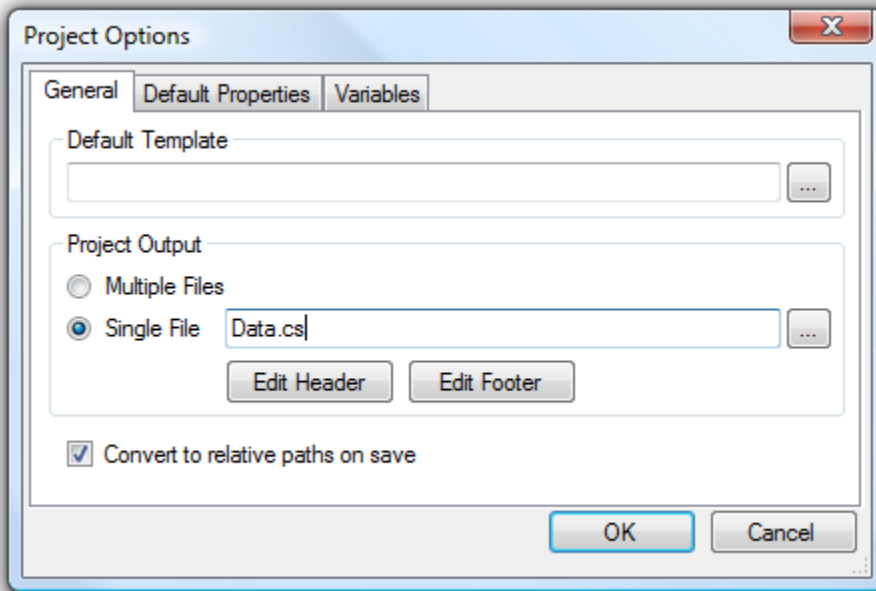
#### Configuring the Project Options

To view a Project Options you must click on the Project Options button. This button is located in the Manage Outputs toolbar as highlighted in green below.



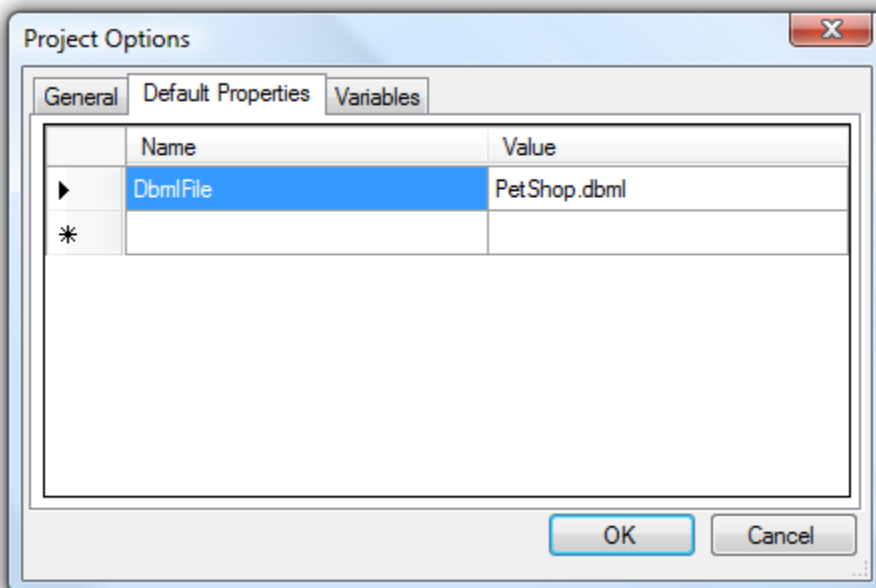
### **Project Output**

The project output option allow the selection between templates controlling output and all the templates output being merged into a single file. When single file output is selected, you can add a header and footer to the output file.



### **Default Properties**

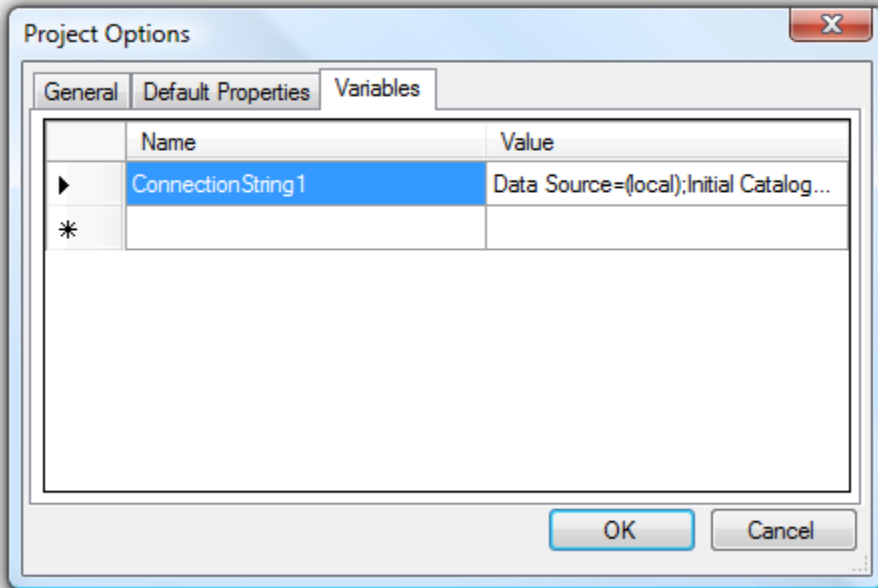
You can configure default properties to use within your CodeSmith Generator Project. These properties are available to all of the templates assigned within your CodeSmith Generator Project. These are especially useful to define properties that are fairly static in nature.



## Variables

The CodeSmith Generator Project supports variables that can be used in the property sets. Variables are an easy way to have a common piece of data that is stored in only one place.

When saving a CodeSmith Project in the Manage Output dialog, the CodeSmith Generator Project will automatically create variables for all the unique connection strings in the project. This allows for easy updating of the connection string for complex projects.



### Variable Usage

When using the Project Options dialog to edit variables, the variables will be placed in the property sets automatically when the project saves. This is done by searching for all values that match the variables value and replacing it with the variable name. The reverse is done when a project is loaded, all variable names are found and replaced with its value.

You can edit the CodeSmith Generator Project manually to place variables as well. The variable for format is its name surrounded by \$(), ie, \$(ConnectionString1).

### Sample Project File

```

<?xml version="1.0"?>
<codeSmith xmlns="http://www.codesmithtools.com/schema/csp.xsd">
  <singleOutput enabled="true" path="Data.cs" />
  <variables>
    <add key="ConnectionString1" value="Data Source=(local);Initial Catalog=PetShop;Integrated
Security=True" />
    <add key="ProviderType"
value="SchemaExplorer.SqlSchemaProvider,SchemaExplorer.SqlSchemaProvider" />
  </variables>
  <defaultProperties>
    <property name="DbmlFile">PetShop.dbml</property>
  </defaultProperties>
  <propertySets>
    <propertySet name="Dbml" template="CSharp\Dbml.cst">
      <property name="IncludeViews">False</property>
      <property name="IncludeFunctions">False</property>
      <property name="EntityBase">LinqEntityBase</property>
      <property name="DisableRenaming">False</property>
      <property name="SourceDatabase">
        <connectionString>$(ConnectionString1)</connectionString>
        <providerType>$(ProviderType)</providerType>
      </property>
      <property name="EntityNamespace">PetShop.Data</property>
      <property name="ContextNamespace">PetShop.Data</property>
    </propertySet>
    <propertySet name="Entities" template="CSharp\Entities.cst">
      <property name="IncludeDataContract">True</property>
      <property name="OutputDirectory">PetShop\PetShop.Data</property>
    </propertySet>
  </propertySets>
</codeSmith>

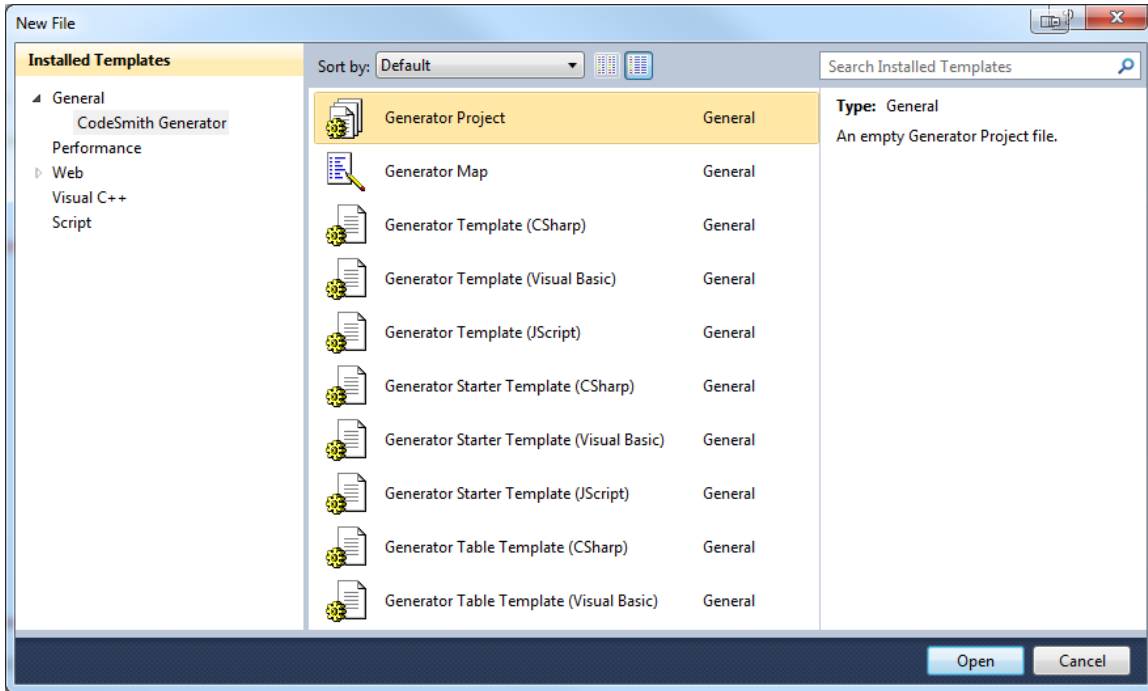
```

## Using a Generator Project inside Visual Studio

The tight integration with Visual Studio allows you to fully manage any [CodeSmith Generator Project](#) right from Visual Studio! This means you can maintain a high Code Generation presence right within Visual Studio and not have to switch applications to run code generation.

### Creating a new CodeSmith Generator Project

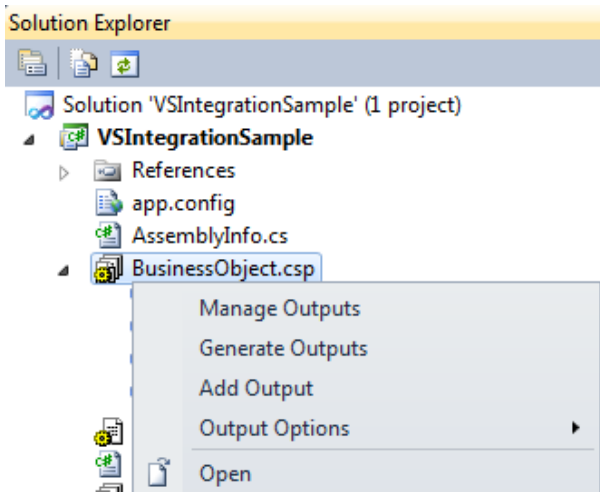
To add a new CodeSmith Generator Project file to your Visual Studio Project, right click on a Project or folder inside of a project inside of Solution Explorer. Then choose File > New Item from the context menu. This will open the Visual Studio New File Wizard. Next, you will want to select CodeSmith Generator under the General Installed Templates node. Doing this will only show you the available CodeSmith Generator Item Templates.



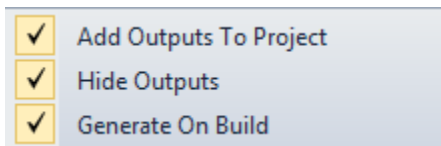
Finally, select **Generator Project** and then select the **Open** button or double click on **Generator Project**.

### Managing a CodeSmith Generator Project From Visual Studio

To manage a CodeSmith Generator Project, you can use the Right-Click context menu of a CodeSmith Generator Project file from the Solution Explorer tool window.




The Output Options sub menu include:

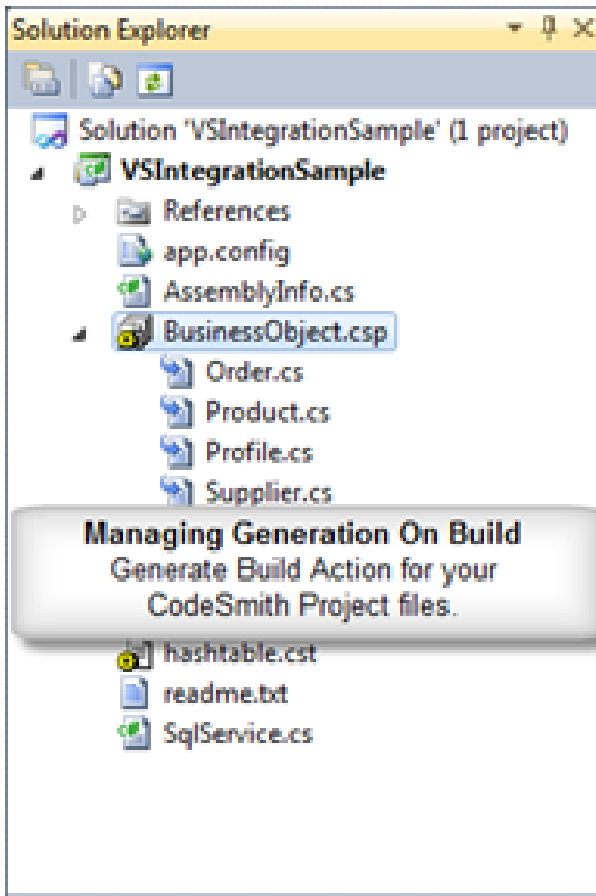


1. **Manage Outputs** - Gives the ability to manage your CodeSmith Generator Project.
2. **Generate Outputs** - Will kick off the generation process to produce outputs configured in your CodeSmith Generator Project.
3. **Add Outputs**
4. **Output Options** - Only available in Visual Studio, and allows you to control the output options after generation.
  - a. **Add Outputs To Project** - Will take any outputs from your CodeSmith Generator Project and include them in your Visual Studio project.
  - b. **Hide Outputs** - Will take any outputs of your CodeSmith Generator Project and make them child nodes, dependant on your CodeSmith Generator Project file (shown in the image below). Hidden nodes can be collapsed, and since much of the time

generated code shouldn't be touched, it's a great way to hide the temptation of other developers attempting to modify the generated code.

- c. **Generate On Build** - A menu driven way to have your templates use Active Generation. You can also specify the BuildAction value in the property sheet of your CodeSmith Generator Project (see the Active Generation section for more information).

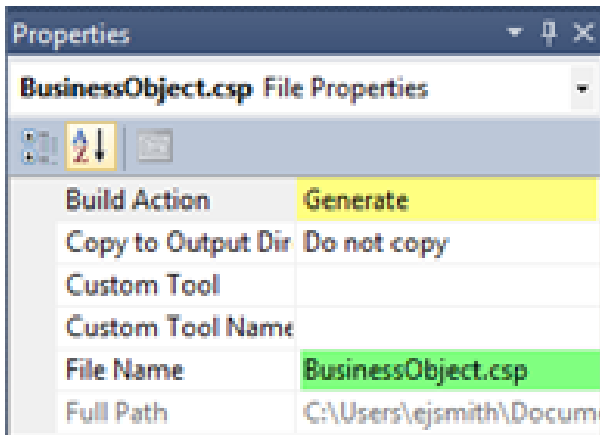
 You can remove the CodeSmith Generator Visual Studio Project dependencies by turning off Generate On Build.



### Active Generation

You can use ActiveGeneration quite easily in your Visual Studio projects now, simply by specifying the BuildAction of your CodeSmith Generator Project.

Setting the **BuildAction = "Generate"** in the properties of your CodeSmith Generator Project file in the Solution Explorer will cause your entire generation process to occur prior to your project building. This means that any Outputs that you have in your CodeSmith Generator Project will be generated and if you want, included in the Visual Studio project.



### Example

We have a BusinessObject.csp CodeSmith Generator Project in our class library Visual Studio project. This CSP has 4 outputs, that generate from the same BusinessObject.cst CodeSmith Generator template.

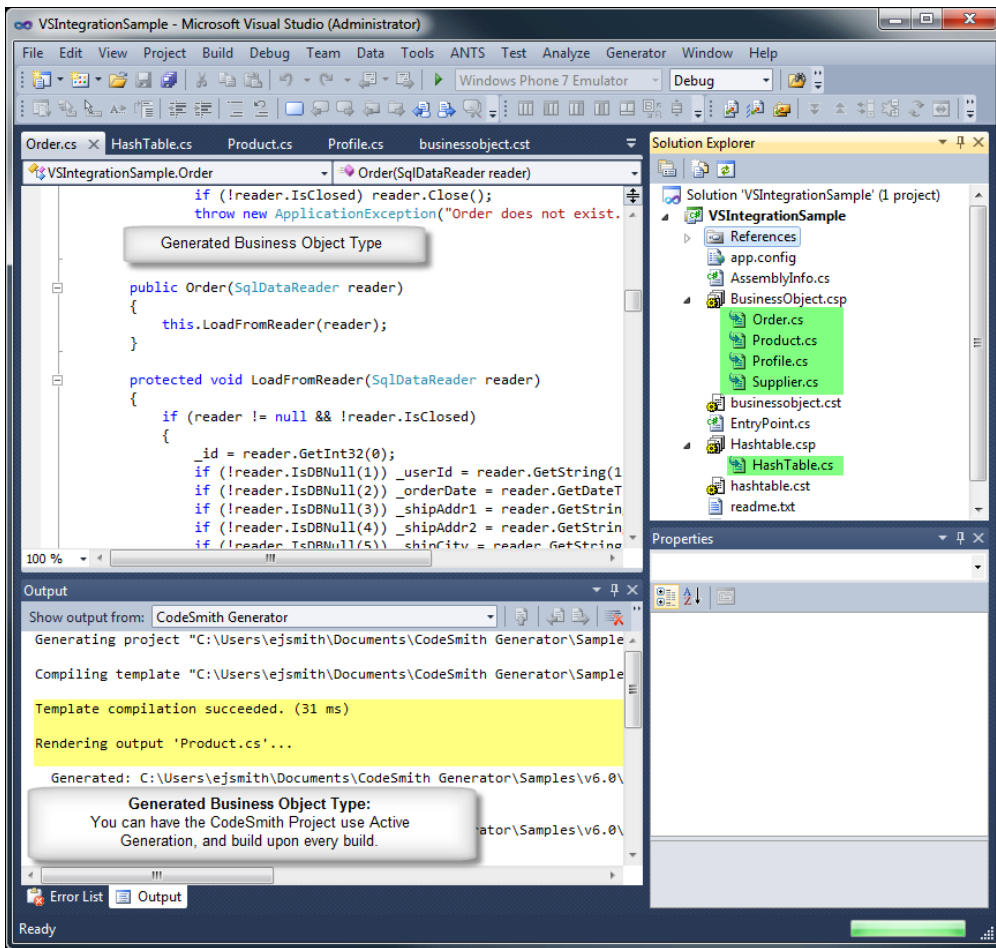
- Order.cs
- Product.cs
- Profile.cs
- Supplier.cs

Looking into the properties of the CodeSmith Generator Project, you can view the Build Action of the file, and there is an option to set it to Generate on Build. Meaning every time you need to tweak your Database meta-data, XML Property meta-data, or CodeSmith Generator template, the changes are picked up in your Visual Studio project the very next time you build.

This enables you to alleviate much of the frequent developer problems with making changes across all of your classes, during the development process.

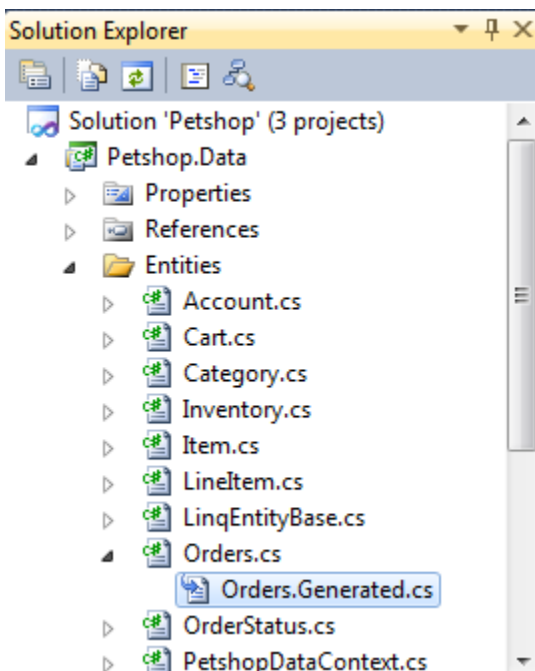
### Output Window Generation Feedback

As you can see depicted in the image below, the business object was generated, and then the build process began. This is a very powerful feature since it allows you to have strong Code Generation integration inside all of your projects in Visual Studio without having to switch to Template Explorer.



## Adding Files to Visual Studio Using DependentUpon Hierarchy

A new overload to the templates `RenderToFile` that will take a parent (DependentUpon) file. This will add metadata to the output file that Visual Studio will use when adding the file to create the hierarchy.



To get your template to support this, you'll need to update the template to use the `RenderToFile` overload that takes a parent file. Next, add the CodeSmith Generator Project to the Visual Studio project and Generate Outputs. CodeSmith Generator will automatically add the outputs to your



Visual Studio project creating the hierarchy.

### Sample Template Code

```
//Create Sub template using the Create method to automatically wire everything up
EntityGeneratedClass entityClass = this.Create<EntityGeneratedClass>();
EntityEditableClass partialClass = this.Create<EntityEditableClass>();

string className = type.Name;

string parentFileName = className + ".cs";
parentFileName = Path.Combine(OutputDirectory, parentFileName);
//Output parent file
partialClass.RenderToFile(parentFileName, false);

string fileName = className + ".Generated.cs";
fileName = Path.Combine(OutputDirectory, fileName);
//Output child (dependent) file linking to parent
entityClass.RenderToFile(fileName, parentFileName, true);
```

## Using Generator Project from Windows Explorer

A *CodeSmith Generator Project* gives you the ability to control all aspects of your CodeSmith Generator project from Windows Explorer, *Template Explorer* or *Visual Studio*. This type of flexibility makes using a CodeSmith Generator Project very useful.

### Creating a new CodeSmith Generator Project

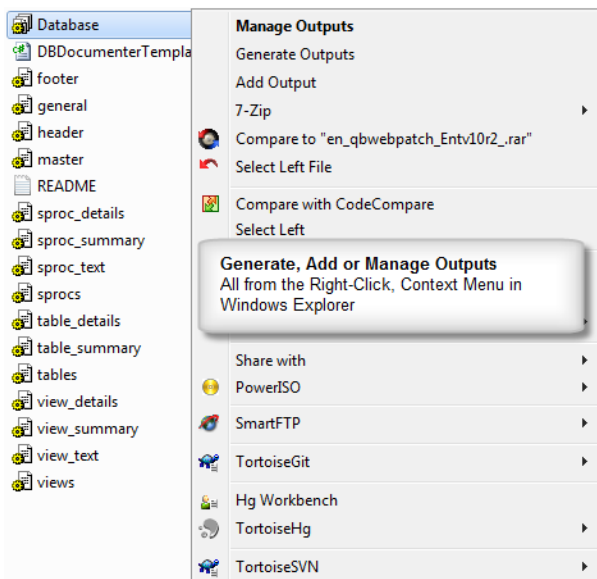
To create a new CodeSmith Generator Project file, right click inside of Windows Explorer and select New -> CodeSmith Generator Project.

### CodeSmith Generator Projects from Windows Explorer or Template Explorer

CodeSmith Generator Projects expose a **right-click** context menu from within Windows Explorer.

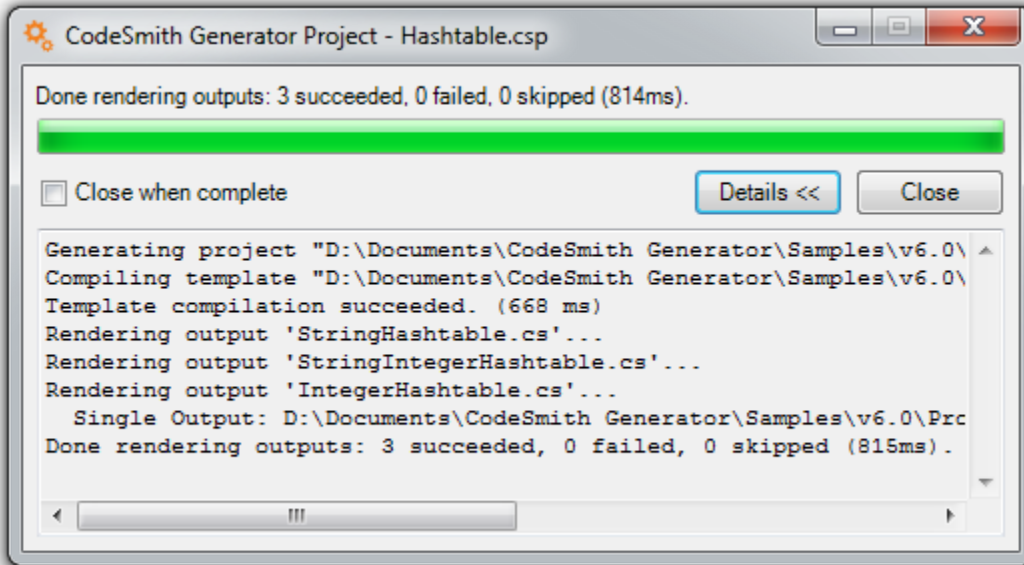
The menu options include:

1. **Manage Outputs** - Gives the ability to manage your CodeSmith Generator Project.
2. **Generate Outputs** - Will kick off the generation process to produce outputs configured in your CodeSmith Generator Project.
3. **Add Outputs**



### Generation Feedback

In Windows Explorer and [Template Explorer](#), you will see a Generation Progress window that will show you progress of your template generation.



**Details:** Toggle Details Button for verbose progress and feedback of your generation process.

**Close when complete:** Closes the window when generation completes.

**Close:** Closes the window.

## Using a Generator Project from MSBuild

*You can create your own custom pre-generation build logic by utilizing the CodeSmith Generator Task within [MSBuild](#). MSBuild tasks help manage the build process within your Visual Studio projects.*

### Why would I want to roll my own when you integrate it with Visual Studio?

There might be times when you need to customize some aspect of the generation process during it's consuming build process. During these time you might have to call CodeSmith Generator from MSBuild using the CodeSmith Generator task that's shipped for you.

### CodeSmith Generator MSBuild Targets

An MSBuild target file is used to define your own tasks during the build process. CodeSmith Generator ships their own targets file which defines all the capabilities of using generating [CodeSmith Generator Projects](#) during the generation process. The CodeSmith.targets file defines how to use the [Generate BuildAction](#) from your Generate Items and also defines how to call CodeSmith with your CodeSmith Generator Project file and run the generation process.

This file can be found at the following location: **C:\Program Files\MSBuild\CodeSmith\CodeSmith.targets** or **C:\Program Files (x86)\MSBuild\CodeSmith\CodeSmith.targets**



When you set the BuildAction to Generate in Visual Studio, you're actually using the CodeSmithGenerate Target and set the project item to use the Generate Build Task.

### Configuration

In order to use the CodeSmith Generation task you must import the CodeSmith.targets file for usage in your MSBuild Project file.

You can import this target in your Visual Studio Projects by using the CodeSmith Generator Import tag in your Visual Studio Project file.

```
<Project DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003 ">
  ...
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets ">
  <Import Project="$(MSBuildExtensionsPath)\CodeSmith\CodeSmith.targets ">
  ...
</Project>
```

## Generating a Generator Project

CodeSmith Generator projects can be run from your MSBuild projects by adding the following line to your project file:

```
<CodeSmith ProjectFile="MyProject.csp" />
```

In this example we will be showing how to call a custom CodeSmith Generator Project that will generate necessary meta-data prior to building in release mode.

### Example

```
<Project DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003 ">
  ...
  <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets ">
  <Import Project="$(MSBuildExtensionsPath)\CodeSmith\CodeSmith.targets ">

  <Target Name="BeforeBuild" Condition="'$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <CodeSmith ProjectFile="GenerateMetaData.csp" >
  </Target>
  ...
</Project>
```

### Usage

The CodeSmith Generator Task exposes a few properties to assist you in your CodeSmith Generator tasks.

**ProjectFiles:** is the property that can accept a single or ";" separated values of names of CodeSmith Generator Projects that will be used for Code Generation.

### Example

```
<CodeSmith ProjectFile="GenerateMetaData.csp;GenerateIndex.csp">
```

**OutputFiles:** is the property that specifies all of the OutputFiles specified in your CodeSmith Generator Project files.

**Verbose:** a boolean property that indicates whether or not to receive verbose messages.

**Debug:** a boolean property that indicates if the templates should be compiled in debug mode.

You can also specify your own templates to run during this custom build process by using the CspFiles tag. This will let you specify your items in an ItemGroup and pass them all into ProjectFiles.

### Example

```

<ItemGroup>
  <CspFiles Include="MyProject.csp" >
  <CspFiles Include="MyOtherProject.csp" >
</ItemGroup>

<Target Name="Build">
  <CodeSmith ProjectFiles ="@(CspFiles)" >
</Target>

```

Find More information on MSBuild Tasks.

## Using a Generator Project from Command-Line

Since a CodeSmith Generator project contains all necessary metadata to run the execution of the most complicated of code generation projects. It makes it possible to call your CodeSmith Generator Project through any batch file, command-prompt, or any application that let's you launch a process with arguments.

The CodeSmith Generator Console Application supports these command-line switches:

```

c:\Administrator: C:\Windows\system32\cmd.exe
C:\Users\Administrator>cs /?
CodeSmith Generator Console v6.0.0.14013
Copyright (c) 2002-2011 CodeSmith Tools, LLC. All rights reserved.

- Generator Console Options -
<file> Project file(s) to be used for output generation or file to
be upgraded with the /upgrade argument.
/property:<string> Assign a property value from the command line
/property:<name>=<value>. Only property types which support
conversion to and from string can be used. (short form /p)
/upgrade:<string> Used to upgrade custom tool and batch format files from
previous versions of CodeSmith to CodeSmith Project files.
This will upgrade the specified file and save it to the
name specified here. /upgrade=<output> (short form /u)
/verbose[+!-] Display verbose messages. (short form /v)
/debug[+!-] Emit debugging information (allows attaching a debugger to
a running template).
/tempfiles[+!-] Keep temporary files (if debug is on then tempfiles will
also be on).
/nologo Suppress generator copyright message.
/resetall Reset all configuration, samples and maps to their defaults.
/resetconfig Reset configuration information to the defaults.
/resetlicense Reset licensing information in case you are having
licensing issues.
/resetsamples Reset samples to the newest default versions.
/resetmaps Reset maps to the newest default versions.
/clearcache Clears all cached template and property entries from the
cache.

Usage samples:
cs sample.csp
cs sample.csp /p:property1=sample
cs customtool.xml /u:newproject.csp
cs batch.xml /u:newproject.csp

C:\Users\Administrator>cs MyCodeSmithProject.csp_

```

### Input Options

<file> Project File to be used for generation

/property:<name>=<value> or /p:<name>=<value> Assign a property value from the command line. Only property types which support conversion to and from string can be assigned in this way.

### Compiler Options

/debug[+!-] Emit (or suppress) debugging information (allows attaching a debugger to a running template)

**/tempfiles[+|-]** Keep (or delete) temporary files (if debug is on then tempfiles will also automatically be on)

## Miscellaneous Options

**/verbose** or **/v** Display verbose messages

**/help** or **/?** Display usage information

**/nologo** Suppress generator copyright message

## Anatomy of a Project File

At a high level, a [CodeSmith Generator Project](#) file manages all of the template properties in a given CodeSmith Generator Project. These files use XML to declaratively manage all of the [CodeSmith Generator template properties](#) for each of your template [Outputs](#) using a **.csp file extension**.

Every CodeSmith Generator Template uses a [Property Sheet](#) that help drive template meta data in your generation process. When the property sheet is saved, it is saved as a Property Set, an XML serialized version of your properties and their values. Saving an XML version enables you easily recover all of the options you designated while configuring your Property Sheet when this file was created.

### Header

The XML Header specifies the current XML Schema Definition for a CodeSmith Generator Project. The CodeSmith Generator node also encapsulates the entire body of the CodeSmith Generator Project XML Files.

```
<?xml version="1.0"?>
<codeSmith xmlns="http://www.codesmithtools.com/schema/csp.xsd">
```

This csp.xsd can be found at `INSTALL DIR/Schemas/csp.xsd`

### Defaults

*Default Template* - You can configure a CodeSmith Generator Project to use Default Template, when configured, each of the Property Sets that do not have a Template assigned to them will automatically use the Default Template to execute and run.

```
<defaultTemplate path="businessobject.cst" />
```

*Default Properties* - You can configure default properties to use within your CodeSmith Generator Project. These properties are available to all of the templates assigned within your CodeSmith Generator Project. These are especially useful to define properties that are fairly static in nature.

```
<defaultProperties>
  <property name="ClassNamespace">CompanyName.RootNamespace</property>
</defaultProperties>
```

### Variables


The CodeSmith Generator Project supports variables that can be used in the property sets. Variables are an easy way to have a common piece of data that is stored in only one place.

You can edit the CodeSmith Generator Project manually to place variables. The variable for format is its name surrounded by `$( )`, ie,

\$(ConnectionString1).

```
<variables>
  <add key="ConnectionString1" value="Data Source=(local);Initial Catalog=PetShop;Integrated
Security=True" />
  <add key="ProviderType"
value="SchemaExplorer.SqlSchemaProvider,SchemaExplorer.SqlSchemaProvider" />
</variables>

<propertySet name="Dbml" template="CSharp\Dbml.cst">
  <property name="SourceDatabase">
    <connectionString>$(ConnectionString1)</connectionString>
    <providerType>$(ProviderType)</providerType>
  </property>
</propertySet>
```

 To learn more on variable usage please click [here](#).

## Property Sets

For every template output you will see the associated PropertySet and it's output. The PropertySet may or may not have a template defined, if it does not, it will use the default template value for generation. The Properties defined within each of the property sets are specific to this Output and are not shared with any other Outputs.

```
<propertySets>
  <propertySet output="Product.cs">
    <property name="SourceTable">
      <connectionString>Data
Source=. \SQLEXPRESS;AttachDbFilename=|DataDirectory|\Petshop.mdf;</connectionString>

      <providerType>SchemaExplorer.SqlSchemaProvider,SchemaExplorer.SqlSchemaProvider</providerType>
      <table>
        <owner>dbo</owner>
        <name>Product</name>
      </table>
    </property>
  </propertySet>

  <propertySet output="Order.cs">
    <property name="SourceTable">
      <connectionString>Data
Source=. \SQLEXPRESS;AttachDbFilename=|DataDirectory|\Petshop.mdf;</connectionString>

      <providerType>SchemaExplorer.SqlSchemaProvider,SchemaExplorer.SqlSchemaProvider</providerType>
      <table>
        <owner>dbo</owner>
        <name>Orders</name>
      </table>
    </property>
  </propertySet>
</propertySets>
```

## Defined Property Sets

Each property set is defined between propertySet nodes as depicted in the image below.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <codesmith xmlns="http://www.codesmithtools.com/schema/csp.xsd">
3   <variables>
4     <add key="ConnectionString" value="Data Source=.;Initial Catal
5     <add key="Provider" value="SchemaExplorer.SqlSchemaProvider,Sc
6   </variables>
7   <propertysets>
8     <propertyset name="Hbms" template="..\..\..\..\Samples\Templat
9     <property name="SourceDatabase">
10      <connectionString>$(ConnectionString)</connectionString>
11      <providerType>$(Provider)</providerType>
12    </property>
13    <property name="IncludeFunctions">True</property>
14    <property name="IncludeViews">True</property>
15  </propertyset>
16  <propertyset name="Entities" template="..\..\..\..\Samples\Tem
17    <property name="SourceDatabase">
18      <connectionString>$(ConnectionString)</connectionString>
19      <providerType>$(Provider)</providerType>
20    </property>
21    <property name="EntitiesDirectory">Entities</property>
22    <property name="ModelsDirectory">Models</property>
23    <property name="GenerateDataContext">True</property>
24    <property name="EntityBaseClass">EntityBase</property>
25  </propertyset>
26 </propertysets>
27 </codesmith>

```

## Using the Console Application

Using the CodeSmith Generator Console Application covers the following sections:

- Incorporating Generator into Your Build Process
- Using a CodeSmith Generator Project from Command-Line
- Basic Console Application Usage
- Handling Input
- Handling Output

## Incorporating Generator into Your Build Process

Using [Template Explorer](#) or the Visual Studio integration interactively to generate code can significantly boost your productivity, but that's only part of what CodeSmith Generator can do for you. In many cases, you'll see even more benefit by incorporating CodeSmith Generator directly into your build process. To facilitate this, CodeSmith Generator includes a [console](#) version that you can call from a CodeSmith Project, a build tool such as NAnt, or a Visual Studio pre-compile task in MSBuild. This helps you ensure that any generated files in your application are always up-to-date.

For example, suppose you are using CodeSmith Generator to automatically create a data access layer and business objects in C# based on selected tables in your database. What happens if you change the schema of those database tables? If you're using CodeSmith Generator manually, you must remember to regenerate the DAL and business objects. But if you've hooked up the CodeSmith Generator Console Application to your build process, the changes in the database schema will automatically be reflected in your final .NET application the next time that you compile the application, without any further effort on your part.

In this section of the help file, you can learn how to:

- Use the CodeSmith Console Application
- Specify template metadata
- Set CodeSmith compiler options

## Basic Console Application Usage

The CodeSmith Generator Console application enables executing CodeSmith Generator Project files from the command line in a very simple manner. Here is an example of executing the PersonArray CodeSmith Project file:

```
cs.exe PersonArray.csp
```

## Handling Input

To make effective use of the CodeSmith Generator Console Application, you must supply the appropriate metadata for the template that you are using as the basis for the generated file. There are two ways that you can do this:

- [By supplying a CodeSmith Generator Project File](#)
- [By supplying properties on the command line](#)

## Specifying Properties on the Command Line

You can specify individual properties on the command line, using the syntax

```
/property:<name>=<value>
```

or

```
/p:<name>=<value>
```

Only property types which support conversion to and from string can be assigned in this way.

You can include multiple instances of the /property switch on the command line to define multiple properties in this way.



If you specify both a property set XML file and a property value on the command line, the property value will override any setting in the property set XML file.

## Handling Output

To see an example of how to handle output check out:

- [Default Output Files in Templates](#)

## Default Output Files in Templates

To use a template's default file name for the output file, specify the `/out:default` command-line switch.

Within a template, you can specify the default output file name by [overriding the `GetFileName` method](#). For example, if your C# language template contains a property named `ClassName`, you might include this code to set the default output file name:

```
<script runat="template">
public override string GetFileName() {
    return ClassName + ".cs";
}
</script>
```


## Using ActiveSnippets

CodeSmith Generator delivers strong integration within [Visual Studio](#) and ActiveSnippets are a driver toward increasing developer productivity. ActiveSnippets, at a high level, are CodeSmith Generator templates with exposure to the entire .Net Framework which you can utilize with a few keystrokes inside of Visual Studio. The output of your



ActiveSnippet will be rendered right where you expanded on the code editor.

You can watch this video tutorial for more information:


 Visual Studio currently offers support for simple template based snippets. This is a great feature for simple snippets of code, however these templates do not contain any advanced logic or access to rich meta-data. For simple snippets such as creating a simple property shell without any logic, we still recommend using the Visual Studio snippet.


## Things to consider before creating an ActiveSnippet

- You have access to the entire .Net Framework, SchemaExplorer, XmlProperty, Custom Assemblies, or any other rich meta-data.
- Complex Objects such as an XmlProperty and SchemaExplorer types can exist as properties in your CodeSmith Template, and similarly act as arguments in an ActiveSnippet.
- An ActiveSnippet can setup default values for arguments that are fairly static in your template.

## Creating an ActiveSnippet Template

The first step in creating an ActiveSnippet is by simply creating a [CodeSmith Generator Template](#). In this example, we'll create a CodeSmith Generator Template that doubles as an ActiveSnippet and is able to generate properties for a given TableSchema in C#.

 This template can be found in Template Explorer under the ActiveSnippets\CSharp or ActiveSnippets\VisualBasic folders.

 For more information on creating a new template please take a look at the [following tutorial](#).

## Requirements

The required output needs to look like this for every column in a table.

```
private int _orderId;
public int OrderId
{
    get { return _orderId; }
    set { _orderId = value; }
}
```

## Writing the Template

When first creating a template, think about the requirements that we defined above and focus and write the template in small steps. This keeps you from getting overwhelmed by writing everything at once. Next, try and think about how you would write the template by outlining some steps in [pseudocode](#). This will make writing the template much easier. Here are a few steps that we used to create this template:

1. Iterate through the Columns (SchemaExplorer.ColumnSchemaCollection) of the template property SourceTable (SchemaExplorer.TableSchema).
2. Create both a field and a property to encapsulate the column.
3. Use a [CodeSmith Generator Map](#) to get the correct type for the field and property.
4. Ensure my field is CamelCased and my Property Name is PascalCased.

Finally, ensure that the template compiles and runs from [Template Editor](#) or [Template Explorer](#).

```

1 <%--
2 Name: Database Table Properties
3 Author: Paul Welter
4 Description: Create a list of properties from a database table
5 --%>
6 <%@ CodeTemplate Language="C#" TargetLanguage="C#" Debug="False" Description="Create a list of
7 <%@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" Category="Context" Descript
8 <%@ Map Name="CSharpAlias" Src="System-CSharpAlias" Description="System to C# Type Map" %>
9 <%@ Assembly Name="SchemaExplorer" %>
10 <%@ Import Namespace="SchemaExplorer" %>
11
12 <% foreach (ColumnSchema column in this.SourceTable.Columns) { %>
13 private <%= CSharpAlias[column.SystemType.FullName] %> _<%= StringUtil.ToCamelCase(column.Nam
14
15 public <%= CSharpAlias[column.SystemType.FullName] %> <%= StringUtil.ToPascalCase(column.Name
16 {
17     get { return _<%= StringUtil.ToCamelCase(column.Name) %>; }
18     set { _<%= StringUtil.ToCamelCase(column.Name) %> = value; }
19 }
20
21 <% } %>

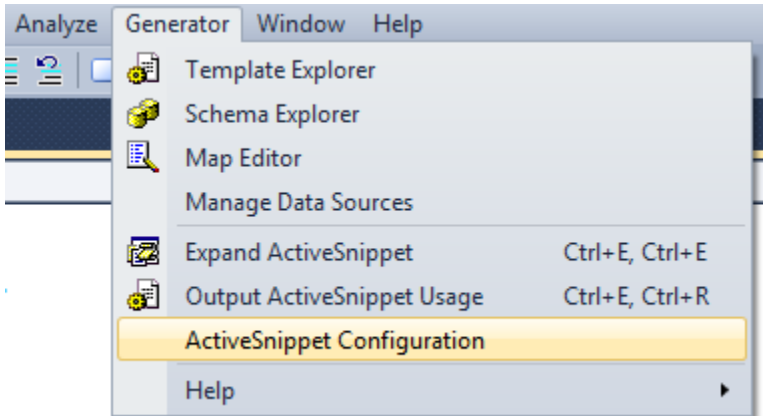
```

Power of Templates:  
Create a Template and use it as an Active Snippet

StringUtil has several common helper

## Visual Studio Integration


The next step is to launch Visual Studio, and explore the options available for using ActiveSnippets within Visual Studio. This will help you get a feel for the integration capabilities for using your ActiveSnippets.



To access various ActiveSnippet features, you can use keyboard shortcuts or use the ActiveSnippet menu items. These are located in the Generator submenu on the right hand side of the Visual Studio menu bar. Once the Generator menu is expanded, you will see the menu items for ActiveSnippet's as shown above. The command keyboard shortcut is located to the right of every menu option that has one configured.

## ActiveSnippet Configuration

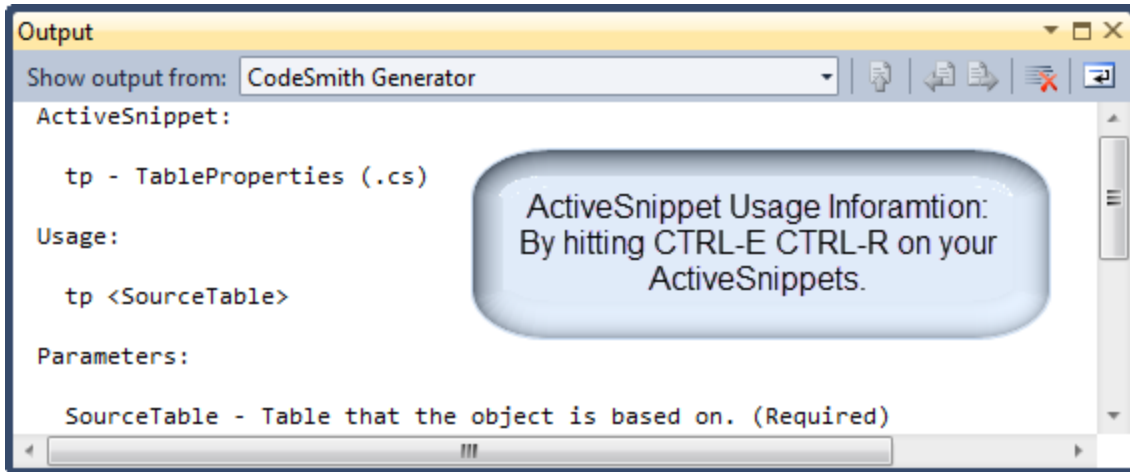
The ActiveSnippet Configuration can be accessed by selecting the ActiveSnippet Configuration menu item located in the Visual Studio Generator sub-menu. The ActiveSnippet Configuration dialog allows you to Add, Remove, Edit or view all ActiveSnippets, Once you've created a template for usage as an ActiveSnippet, you must add the ActiveSnippet which maps to a CodeSmith Generator Template.

 ActiveSnippets must be **configured** inside Visual Studio in order to be used.

Find detailed information on [Configuring an ActiveSnippet](#).

## Output ActiveSnippet Usage

ActiveSnippet usage information can be obtained through the CodeSmith Generator [Output Window](#). CodeSmith attempts to find the ActiveSnippet usage information using the context of the current line with focus. To display the output usage for configured ActiveSnippets you can select the **Output ActiveSnippet Usage** menu item or press **Ctrl+E, Ctrl+R**.



### Notable Information

- Executing Usage with no alias on the editor will display all ActiveSnippets.
- Executing Usage using part of the prefix, will display a list of all ActiveSnippets starting with that prefix
  - For Example, using "t" by itself will show a list of all ActiveSnippets beginning with a "T".

### Expanding an ActiveSnippet

Attempts to execute the ActiveSnippet using the context of the current line with focus. If there is an ActiveSnippet configured and no errors, CodeSmith Generator will attempt to find the ActiveSnippet by Alias or by Name. If the ActiveSnippet is found, CodeSmith Generator will compile the template if not compiled, and then execute the template with the given arguments. The template output of the ActiveSnippet will be placed on the editor control of Visual Studio.

### Syntax

Calling an ActiveSnippet is easy. Once [configured](#), you simply have to enter the alias or name along with any argument parameters. Once you have defined the active snippet you want to expand, you just need to select the **Expand ActiveSnippet** menu item or press **CTRL-E, CTRL-E**.

**i** If you are unsure about an ActiveSnippets usage is, you can select the **Output ActiveSnippet Usage** menu item or press **Ctrl+E, Ctrl+R**.

**i** By default the shortcut for Expanding an ActiveSnippet is **CTRL-E, CTRL-E**.

### Example

In the example below we will execute an ActiveSnippet with the name `tp` and pass it one argument parameter.

```
tp Petshop.dbo.Orders
```

- You can also access an ActiveSnippet by referring to it's full name.
- You can use complex objects, such as a TableSchema by referring to it's fully qualified name `Petshop.dbo.Orders`

to execute this active snippet we will select the **Expand ActiveSnippet** menu item or press **CTRL-E, CTRL-E**. The below screenshot shows the code which was generated by this ActiveSnippet.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace CSharpCodeGeneratorSample
6 {
7     class Sample
8     {
9         private int _orderId;
10
11         public int OrderId
12         {
13             get { return _orderId; }
14             set { _orderId = value; }
15         }
16
17         private string _userId;
18
19         public string UserId
20         {
21             get { return _userId; }
22             set { _userId = value; }
23         }
24
25         private System.DateTime _orderDate;
26
27         public System.DateTime OrderDate
28         {
29             get { return _orderDate; }
30             set { _orderDate = value; }
31         }
32
33         private string _shipAddr1;
34
35         public string ShipAddr1

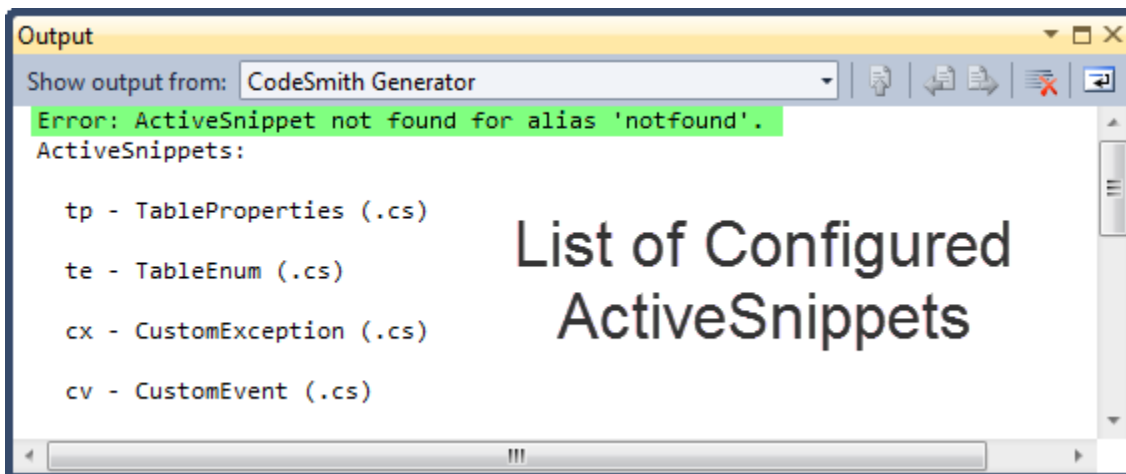
```

Expanding Active Snippets:  
Expand: CTRL-E + CTRL-E

Once expanded you now see the power of CodeSmith Templates as an ActiveSnippet and increase your developer productivity exponentially.

### Notable Information

- If there is an error executing an already configured ActiveSnippet, usage information on the discovered ActiveSnippet will be presented. This shows the ActiveSnippet along with all of the arguments for that template.
  - Template is not valid. SourceTable is required. tp - TableProperties (.cs)
- If CodeSmith Generator can not find the desired ActiveSnippet by name or by configured alias, then a full list of all available ActiveSnippets will be presented in the CodeSmith Generator Output Window.



### ActiveSnippet Configuration

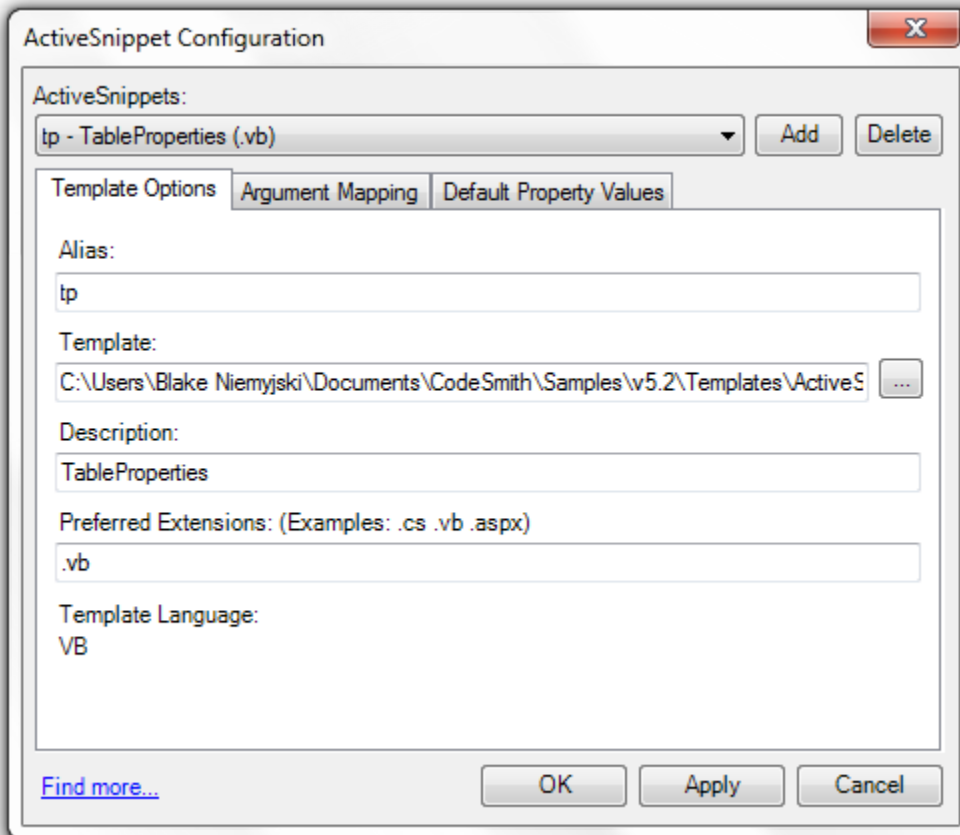
The ActiveSnippet Configuration can be accessed by selecting the ActiveSnippet Configuration menu item located in the Visual Studio Generator sub-menu. The ActiveSnippet Configuration dialog allows you to Add, Remove, Edit or view all ActiveSnippets, Once you've created a template for usage as an ActiveSnippet, you must add the ActiveSnippet which maps to a CodeSmith Generator Template.

## Adding a new ActiveSnippet

The first step to creating a new ActiveSnippet is to click the **Add button**, and use the Template Chooser Window to browse to the CodeSmith Generator template that will be serving as your ActiveSnippet. Once you select the template, you must configure at minimum the Template Options for the template.

### Template Options

The template options tab holds the mapping information about your CodeSmith Generator template. This information is required to allow you to have access to this template from within Visual Studio.

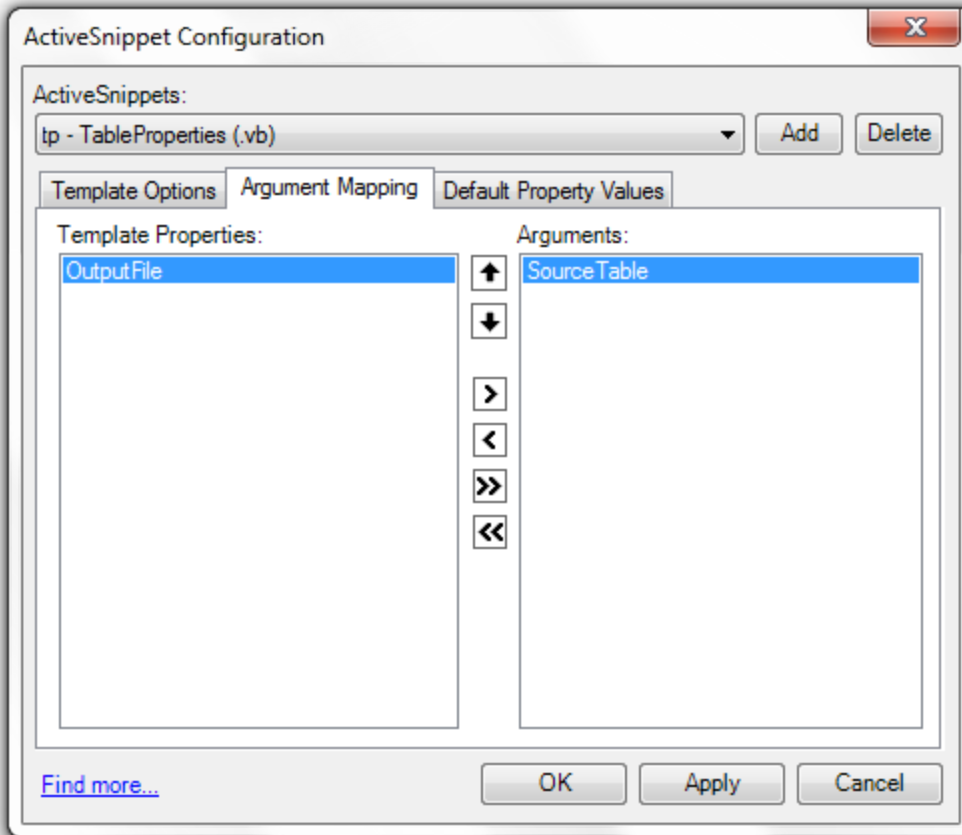


**Template:** The full path to the CodeSmith Generator template.

**Alias:** The ActiveSnippet Alias is the command to be used as the alias representing the selected CodeSmith Generator template.  
**Description:** Specifies the friendly name description of the ActiveSnippet. It will be shown during the Output Usage information.  
**Target Extensions:** The template target language is used as a hint for ActiveSnippets that have the same name.  
**Template Language:** The template language shows the selected CodeSmith Template Output Language.

### Argument Mapping

Configuring the arguments for an ActiveSnippet is a powerful feature because it does not force you to have to always pass all properties in for the selected CodeSmith Generator Template as arguments.

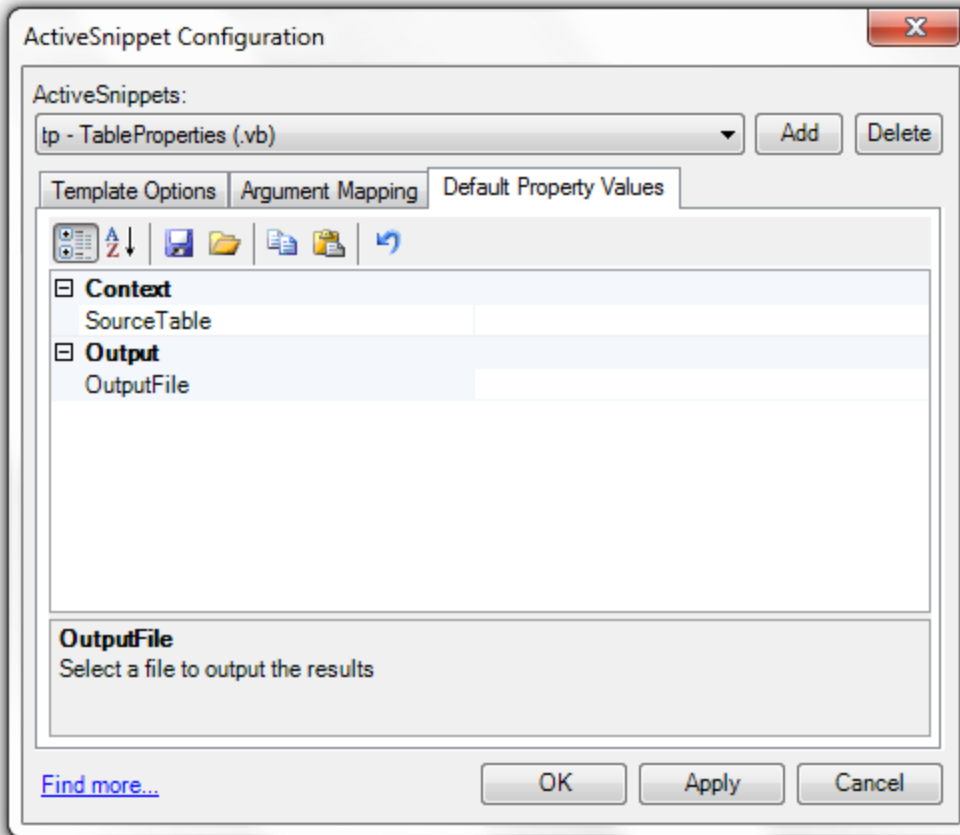


This tab is a dual pane select box which shows all template properties to the left, and all arguments to the right. Control arrows are used to move properties to and from the Arguments window. The up and down arrows are used to setup the argument order of the ActiveSnippet.

Template Properties Box: A window of all the property names in a template. Arguments Box: A window of all the arguments that will be required to use your ActiveSnippet

### Default Property Values

The Default Property Values tab will allow you to enter possible default values for your CodeSmith Generator Templates' Property Sheet. This will show all the properties for the ActiveSnippet based CodeSmith Generator Template.



Values that you want to be required properties for ActiveSnippet arguments would not be filled in as a default value.

## Basic Template Syntax

Basic Template Syntax covers the following sections:

- [The CodeTemplate Directive](#)
- [Including Comments](#)
- [Declaring and Using Properties](#)
- [Escaping ASP.NET Tags](#)
- [The CodeSmith Generator Objects](#)

## The CodeTemplate Directive

Every CodeSmith Generator template must start with a CodeTemplate directive. Every template must contain precisely one CodeTemplate directive. The only thing that can appear before the CodeTemplate directive in the template is one or more [comments](#).

The CodeTemplate directive is the only required directive and is used to specify the general properties of the template, such as the language that the template is written in and the description. For example, here's the CodeTemplate directive from the SortedList.cst sample template:

```
<%@ CodeTemplate Language="VB" TargetLanguage="VB" Description="Generates a strongly-typed collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index." %>
```

This directive specifies that the template uses Visual Basic as its own code-behind language, and that it produces VB output. It also includes a description of the purpose of the template.

### CodeTemplate Directive Attributes

There are seven attributes that you can supply to the CodeTemplate directive. The Language parameter is required; all of the rest are optional.

## Language

The Language attribute specifies what language will be used to write the template. Possible values for this attribute are:

- C# to author the template in C#
- JS to author the template in JScript
- VB to author the template in Visual Basic .NET

## TargetLanguage

The TargetLanguage attribute is used to specify the output language of the template. You can use any string you like for the attribute; CodeSmith Generator doesn't use it in any way to generate the template's output. The [Template Editor](#) also uses this attribute to determine how to syntax highlight the static content of a template.

## Description

The Description attribute is used to describe your template in a general way.

## Inherits

By default all CodeSmith Generator templates inherit from [CodeSmith.Engine.CodeTemplate](#). This class provides the basic functionality of the template; much like the Page class provides the basic functionality of an ASP.NET page. The Inherits attribute can be used to specify that a template inherits from a different class. However, any class that a template inherits from must inherit, directly or indirectly, from CodeSmith.Engine.CodeTemplate. CodeSmith Generator must also be able to find this class. To ensure this, you must either supply an [Assembly](#) directive pointing to the assembly that contains the class, or a Src attribute that points to the source code for the class.

For an example of using template inheritance, see the BaseTemplates sample project included with your CodeSmith Generator installation. This project defines two new template classes, OutputFileCodeTemplate which inherits directly from CodeTemplate and SqlCodeTemplate which inherits from OutputFileCodeTemplate. To base a new template on SqlCodeTemplate you could include these directives at the top of your template:

```
<%@ CodeTemplate Language="CS" Inherits="CodeSmith.BaseTemplates.SqlCodeTemplate" %>
<%@ Assembly Name="CodeSmith.BaseTemplates" %>
```

Having done this, all of the helper methods defined in the OutputFileCodeTemplate and SqlCodeTemplate classes, such as GetSqlDbType(), IsUserDefinedType(), GetSqlParameterStatements(), and many more, are available to your template. Template inheritance thus provides a good way to reuse tested utility methods across multiple templates without cut-and-paste duplication of code.



The BaseTemplates sample can be found in your extracted samples ( ...\CodeSmith Generator\Samples\<VERSION>\Projects\CSharp\BaseTemplates )

## Src

The Src attribute allows you to include functionality from another class in your template by dynamically compiling that class file as part of your template. Set the value of the attribute to point to the source file of the class that you want to include in the template. You use the Src attribute to enable the [CodeSmith Generator code-behind model](#).

## Debug

The Debug attribute is used to determine whether or not debug symbols should be included in the generated assembly. Click [here](#) to learn more about debugging.

## LinePragmas

The LinePragmas attribute is used to determine whether or not line pragmas are generated during template compilation. When this attribute is set to True, template errors will point to the template source code. If it is set to False, then template errors will point to the compiled source code.

## ResponseEncoding

The ResponseEncoding attribute is used to set the encoding for the template content and its outputs. The ResponseEncoding attribute supports values from the System.Text.Encoding.GetEncoding method. By default, the encoding is set to ASCII.

## OutputType

The OutputType attribute is used to set the output type of the template. The following values can be used:



- **Normal** - This is the default setting and will cause the output of the template to be written to the normal Response stream.
- **Trace** - This setting will cause the output of the template to be written to the Trace object.
- **None** - *This setting will cause the template to not output anything.* This is useful in a master template scenario where the template just calls other templates and outputs those to files and the master template itself doesn't output anything.

### **NoWarn**

Comma-delimited list of the warning ID numbers that the template compiler should suppress. These are standard C# / VB compiler warning ID numbers.

### **ClassName**

The ClassName attribute is used to specify the class name of the compiled template. This attribute should be defined when using partial code-behinds.

### **Namespace**

The Namespace attribute is used to specify the namespace for the compiled template. This attribute should be defined when using partial code-behinds.

### **Encoding**

The Encoding attribute allows you to define the encoding the current [template document](#) will be saved as. The default encoding of a template document is UTF-8.

### **ResponseEncoding**

The ResponseEncoding attribute allows you to define the encoding the [generated document](#) will be saved as. The default encoding of a generated document is UTF-8.



If a generated document already exists on disk and the regenerated documents contents match exactly, the documents encoding will not be changed.

## **Including Comments**

To include comments in a template, surround them with `<%--` and `--%>` markers. Comments may span multiple lines. For instance, this comment will have no effect on the template's output:

```
<%--
Name: TestHarness.cst
Description: Generates a standard test harness for an object
--%>
```



Inside of a script block, use the commenting syntax of the template's language. For instance, if your template is written in C#, comments in script blocks should be prefaced with `//` or `/* commented */`.



**To include a comment in a template's output, treat it like any other string. The comment in this template fragment will be copied directly to the generated code:**

```
<%@ CodeTemplate Language="VB" TargetLanguage="VB">
' This class generated by CodeSmith Generator
```

## **Declaring and Using Properties**

The key to making templates flexible and useful is to define properties or metadata. CodeSmith Generator uses properties to customize the generated code. When the user opens a template in Template Explorer, they must [supply values](#) for all of the required properties defined in the template before they can generate the code. Note that properties can be defined as [optional properties](#), in which case the user need not supply a value before generating code.

Properties are specified in the template using a [Property directive](#). For example, this directive specifies a property named Key which accepts a string value:

```
<%@ Property Name="Key" Type="System.String" %>
```

The value that the user enters for the Key property will be inserted into the template output any place that the property name appears surrounded by the special characters <%= %>. For example, consider this template:

```
<%@ CodeTemplate Language="VB" TargetLanguage="VB">
<%@ Property Name="Key" Type="System.String" %>
' The key is <%= Key %>
```

If the user enters Ham for the value of the Key property, then the output of this template would be:

```
' The key is Ham
```

Properties in CodeSmith Generator can be simple or very complex. You can define [enumerated properties](#) that allow the user to choose from a predefined selection of values. You can use CodeSmith Generator's [SchemaExplorer](#) to fill properties from database objects. You can create properties based on the contents of an [XML file](#), or even define your own [custom properties](#) complete with custom dialog boxes for user property editing.

## Property Directive

To declare a property, you use a Property directive. For example, this directive defines a property named ClassName of type System.String:

```
<%@ Property Name="ClassName" Type="System.String" Category="Context" Description="The name of the
class to be generated." %>
```

### Property Directive Attributes

The Property directive has nine possible attributes. The Name and Type attributes are required, and the other attributes are optional.

#### Name

The Name attribute is used as the name of the property when it is displayed on the template's property sheet in CodeSmith Explorer. This is also the variable name that is used to store the value of the property within the template. This must be a legal variable name within the template's language. For example, if the template uses C# as its language, then the name must follow the rules for C# variables.

#### Type

The Type attribute specifies the .NET type of the property. This parameter can be any .NET data type, though for complex types you may need to specify an Editor attribute to allow the user to successfully supply a value for the property.



For scalar types, you must use Base Class Library types such as String or Int32 rather than language-specific types such as string or int.

#### Default

The Default attribute is used to set the default value for this property. If you omit this attribute, then CodeSmith Generator does not supply a default value for the property.

#### Category

The Category attribute specifies what category this property should appear under in the CodeSmith Explorer property sheet. If you omit this attribute, CodeSmith Generator will place the property in a category named Misc.

#### Description

The Description attribute supplies descriptive text to be displayed at the bottom of the property sheet when this property is selected.

Optional

The Optional attribute specifies whether or not this property is optional. If a user does not specify a parameter that is not optional then CodeSmith Generator will not let them proceed. A value of true means that a value for the property is not required, and a value of false means that a value for the property is required.

Editor

The Editor attribute specifies the GUI editor that will be used in the property grid for this property. This is equivalent to placing an [EditorAttribute] on a code property.

EditorBase

The EditorBase attribute specifies the base type for the editor. If none is specified, then UITypeEditor is assumed.

Serializer

The Serializer attribute specifies the IPropertySerializer type to use when serializing the property's values. This is equivalent to using a [PropertySerializerAttribute] on a code property.

OnChanged

The OnChanged attribute specifies the event handler to fire when the property value changes.

DeepLoad

The DeepLoad attribute is only used on [SchemaExplorer](#) objects. When set to true, SchemaExplorer will grab all your schema information in advance saving make multiple round trips back to your database.

### **Declaring a Property From the CodeBehind**

Declaring a property from code is essentially like creating a property in any class. The most notable options using Attributes to help you describe your property, it's location, and it's editor.

**Example:**

```
private string aliasFilePath;

    [Editor(typeof(System.Windows.Forms.Design.FileNameEditor),
    typeof(System.Drawing.Design.UITypeEditor))]
    [Category("01. General")]
    [Optional]
    [DefaultValue("")]
    [Description("Optional File Path to a table/object alias file.")]
    public string AliasFilePath
    {
        get {return this.aliasFilePath;}
        set {this.aliasFilePath = value;}
    }
```

**EditorAttribute** - Specifies which editor to use in the Property Sheet.

**CategoryAttribute** - Specifies a Property Sheet group this option belongs to.

**OptionalAttribute** - If declared the property will be marked as optional.

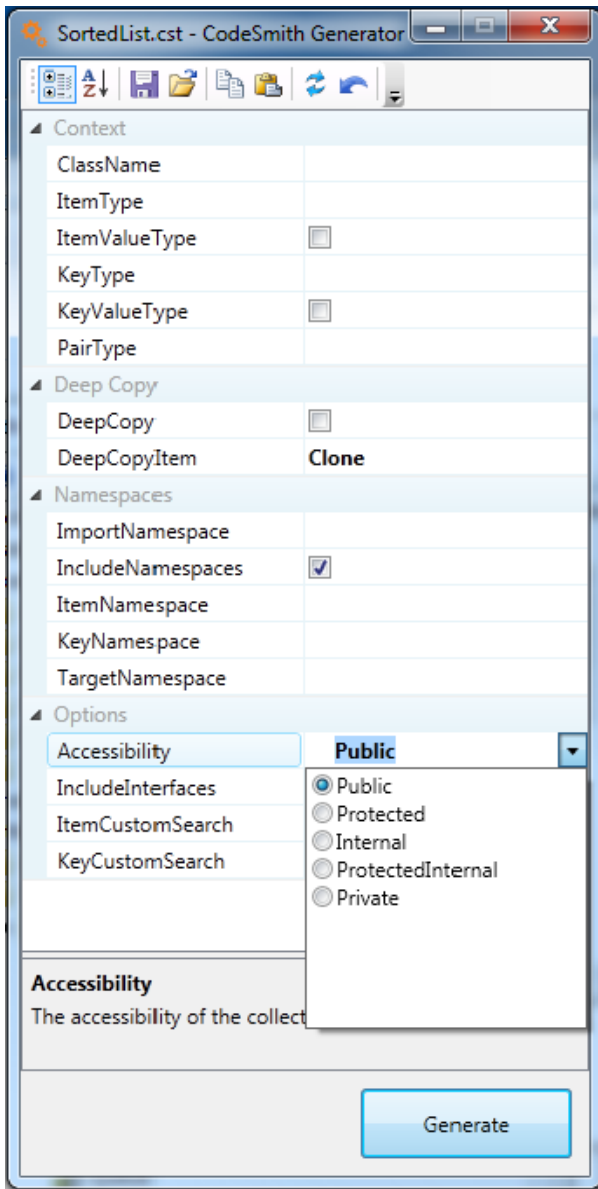
**DefaultValue** - Specify the default value for the property.

**DescriptionAttribute** - Used to create a description for the selected property.

**CodeTemplatePropertyAttribute** - Has been deprecated.

### **Declaring an Enumerated Property**

Sometimes it's convenient to have a property that limits the user to selecting from a fixed set of choices. For instance, in the SortedList.cst template, the Accessibility property controls the accessibility of the generated class:



To accomplish this, you need to take two steps. First, define an enumeration in a <script> block in your code:

```

<script runat="template">
Public Enum AccessibilityEnum
    [Public]
    [Protected]
    [Friend]
    [ProtectedFriend]
    [Private]
End Enum
</script>

```

Second, the Property directive should point to the enumeration:

```

<%= Property Name="Accessibility" Type="AccessibilityEnum" Category="Options" Description="The
accessibility of the class to be generated." %>

```

That's all there is to it!

## Property Validation

Because you can define a property as being [optional](#), you may want to validate the property in your template to determine whether or not the user has entered a value. For example, you might want to allow generating a class either with a namespace declaration or without, at the user's option. To do this, you would first define an appropriate optional property:

```
<%@ Property Name="ClassNamespace" Type="System.String" Optional="True" Category="Context"
Description="The namespace that the generated class will be a member of." %>
```

In your template, you can check to see whether there's a value in this property at runtime. If so, you want to output the appropriate namespace declaration. If you're using C# as your template language, you'd do that like this:

```
<% if (ClassNamespace != null && ClassNamespace.Length > 0)
{ %>namespace <%= ClassNamespace %>{<% }
%>
```

If your template is using VB, the equivalent code is:

```
<% If Not ClassNamespace Is Nothing AndAlso ClassNamespace.Length > 0 Then %>
Namespace <%= ClassNamespace %>
<% End If %>
```

## Escaping ASP.NET Tags

If you're building ASP.NET code with CodeSmith Generator, you'll run into the problem that the `<%` tags that you want to output to your ASP.NET code are interpreted by CodeSmith Generator as CodeSmithtags instead. The solution is to escape the starting tags, replacing `<%` with `<%%`. This will be replaced with `<%` in the output, and not seen by CodeSmith Generator as an opening script tag.

## The CodeSmith Generator Objects

Behind the scenes, the CodeSmith Generator engine works by manipulating a rich object model. That object model is exposed for your templates to work with as well. In this section, we'll explore some of the things that you can do with the CodeSmith Generator objects:

- The [CodeTemplate](#) object
- The [Progress](#) object
- The [CodeTemplateInfo](#) object

### The CodeTemplate Object

The `CodeTemplate` class represents your entire template as it's being processed by CodeSmith Generator. You can work with a `CodeTemplate` object to interact directly with the CodeSmith Generator engine. For example:

- Use the [GetFileName](#) method to specify the default output file name for a template
- Use the [Render](#) method to render the output of the template
- Use [events](#) of the object to insert your own code into the CodeSmith Generator processing cycle
- Use the [Response](#) property to write directly to the template output

### Overriding the GetFileName Method

CodeSmith Generator uses the `GetFileName` method to provide a default output file name for the template when it's called from the CodeSmith Generator [Console Application](#), [Template Editor](#) or [Master Template](#). This is also used in CodeSmith Generator as the default file name if you save the output of a template, and anywhere else that CodeSmith Generator needs to assign a filename to the output of your template. You can override this method in your code when you want to build the default file name based on property input or other factors.

For example, if your C# language template contains a property named `ClassName`, you might include this code to set the default output file name:

```

<%@ Template Language="C#" TargetLanguage="Text" %>
<%@ Property Name="ClassName" Type="System.String" Default="ClassName" %>

This template shows off how to override the GetFileName method.

<script runat="template">
public override string GetFileName()
{
    return ClassName + ".cs";
}
</script>

```

### Example

Using the template defined below we will show off how changing the database table we generate off of, changes the the file path that the template is rendered to.

```

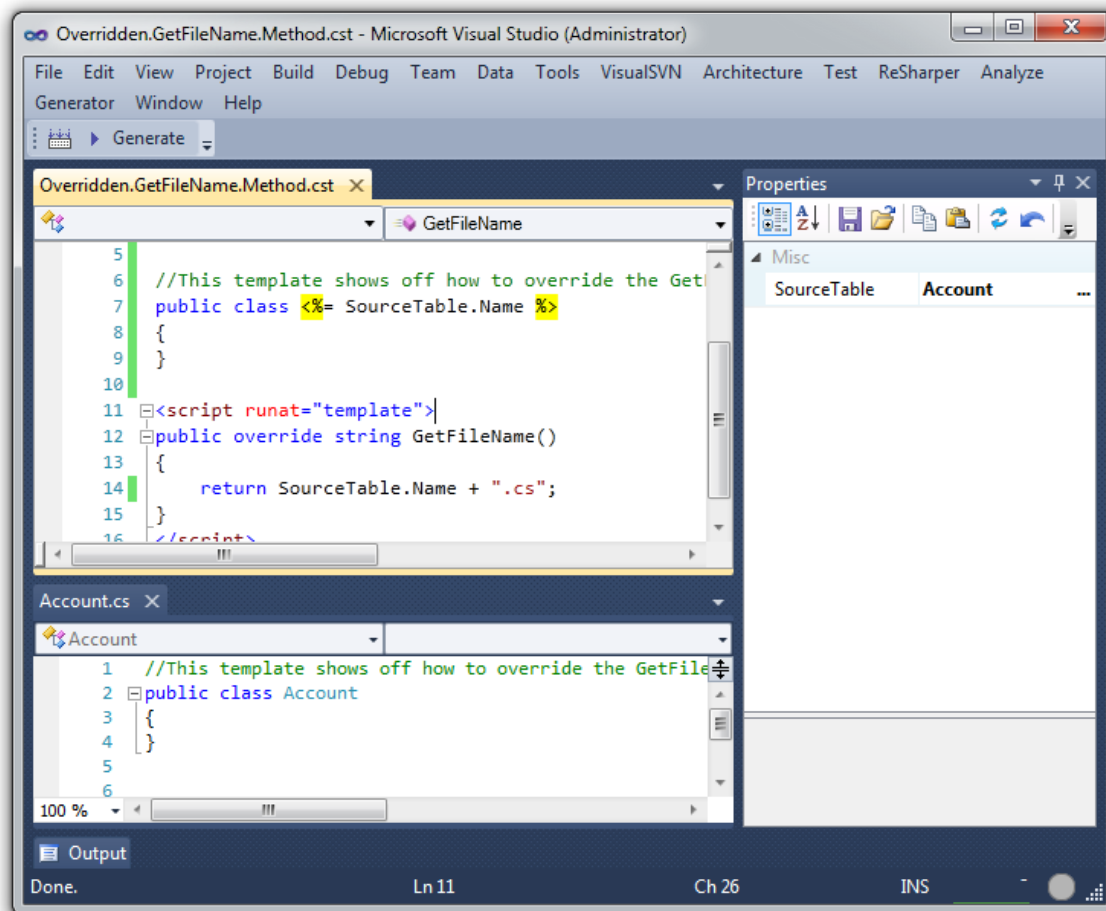
<%@ Template Language="C#" TargetLanguage="C#" %>
<%@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" %>
<%@ Assembly Name="SchemaExplorer" %>
<%@ Import Namespace="SchemaExplorer" %>


//This template shows off how to override the GetFileName method using a Database Table.
public class <%= SourceTable.Name %>
{
}

<script runat="template">
public override string GetFileName()
{
    return SourceTable.Name + ".cs";
}
</script>

```

The first step is to create the template above or download the one attached below. Next, choose a table your wish to generate against by configuring the SourceTable Property via the PropertyGrid. In the screenshot below we are choosing a random database table called Account. Finally, we click Generate to render the templates contents to a file called Account.cs file name.



 You can download this template by clicking [here](#).

### Overriding the ParseDefaultValue Method

You may sometimes want to define a property with a default value that cannot be automatically converted from a string. In this case, you'll need to override the *ParseDefaultValue* method in your template to handle parsing the default value from the template and assigning it to the property. This method is called by CodeSmith Generator for each property in the template, and gets passed the property and the default value string from the template. If you override the method, you can insert whatever custom logic you like to assign values to the property that you care about, while passing other properties to the base *ParseDefaultValue* method.

### Overriding the Render Method

The *CodeTemplate.Render* method is where CodeSmith Generator does the actual work of combining metadata with your template to create the template's output. You can override this method if you want to modify the way that CodeSmith Generator ultimately handles that output. For example, overriding this event allows you to write your template's output to multiple destinations instead of just to the default output window. Here's a template that outputs some text to two files at the same time, as well as to CodeSmith's default output window:

```

<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="AddTextWriter Demonstration." %>
<%@ Import Namespace="System.IO" %>
//This template demonstrates using the AddTextWriter method
//to output the template results to multiple locations concurrently.
<script runat="template">
public override void Render(TextWriter writer)
{
    StreamWriter fileWriter1 = new StreamWriter(@"C:\test1.txt", true);
    this.Response.AddTextWriter(fileWriter1);

    StreamWriter fileWriter2 = new StreamWriter(@"C:\test2.txt", true);
    this.Response.AddTextWriter(fileWriter2);

    base.Render(writer);

    fileWriter1.Close();
    fileWriter2.Close();
}
</script>

```



Don't omit the call to the `base.Render` method. If you forget this, then you won't get the default output!



You also have access to the default `TextWriter` if you override the `Render` method. This means that you can write your own headers or other additional information directly to the output along with the template's output.

## Template Events

The `CodeTemplate` object provides three events that you can use to insert logic during the template processing cycle:

- The `OnInit` event fires when the template instance is created
- The `OnPreRender` event fires just before the template is rendered
- The `OnPostRender` event fires just after the template is rendered
- The `OnPropertyChanged` event fires after a template property has changed.

### The `OnInit` Event

The `OnInit` event fires when the CodeSmith Generator engine creates an instance of your template. You can override this event to perform any necessary setup tasks for your template. For instance, suppose your template uses an [additional `TextWriter`](#) to send a copy of its output to a socket on a remote computer via the Internet. You could override the `OnInit` event to check for Internet connectivity, and warn the user that the template will not succeed if you can't find an open Internet connection when the template is instantiated.

### The `OnPreRender` Event

The `OnPreRender` event is fired just before the CodeSmith Generator engine merges metadata with your template to produce the template's output. One use for this event is to perform "sanity checks" on metadata entered by the user. You could, for example, check that a date entered was within an acceptable range, and change it to the earliest or latest acceptable date if it is not.



Although you can modify metadata in the `OnPreRender` event, you cannot prevent the template from being rendered.

### The `OnPostRender` Event

The `OnPostRender` event is fired after CodeSmith Generator has merged metadata with your template to produce the output. You can use this event to perform any additional processing you would like after CodeSmith Generator has finished its job. For example, the `StoredProcedures.csst` sample template included with CodeSmith Generator uses this event to [autoexecute the generated SQL script](#):



```
protected override void OnPostRender(string result)
{
    if (this.AutoExecuteScript)
    {
        // execute the output on the same database as the source table.
        CodeSmith.BaseTemplates.ScriptResult scriptResult =
CodeSmith.BaseTemplates.ScriptUtility.ExecuteScript(
            this.SourceTable.Database.ConnectionString,
            result,
            new System.Data.SqlClient.SqlInfoMessageEventHandler(cn_InfoMessage));

        Trace.Write(scriptResult.ToString());
    }

    base.OnPostRender(result);
}
}
```

### The OnPropertyChanged Events

The *OnPropertyChanged* event is fired after a template property has been modified. You can use this event to perform any additional processing or validation for the property that has been changed.

```
<script runat="template">
protected override void OnPropertyChanged(string propertyName)
{
    Response.Write(propertyName + "has changed");
    base.OnPropertyChanged(result);
}
</script>
```

### The Response Property

The *Response* property of the *CodeTemplate* object returns an instance of the *CodeTemplateWriter* class. This object represents the actual response stream for the template output. You can write to the stream programmatically using this property, thus inserting your own output directly into the generated template. For example:

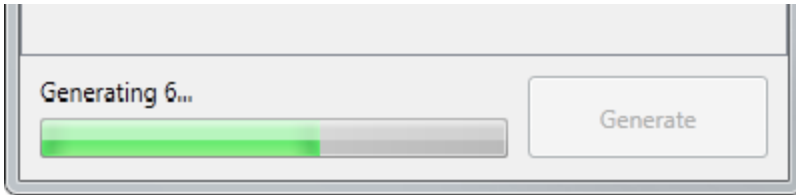
```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="This template demonstrates
writing directly to the Response property" %>
<% RenderDirect(); %>
<script runat="template">
public void RenderDirect()
{
    Response.WriteLine("Written directly to the Response property.");
    Response.WriteLine("Hello " + System.Environment.UserName + "!");
}
</script>
```

Useful methods of the *CodeTemplateWriter* class include:

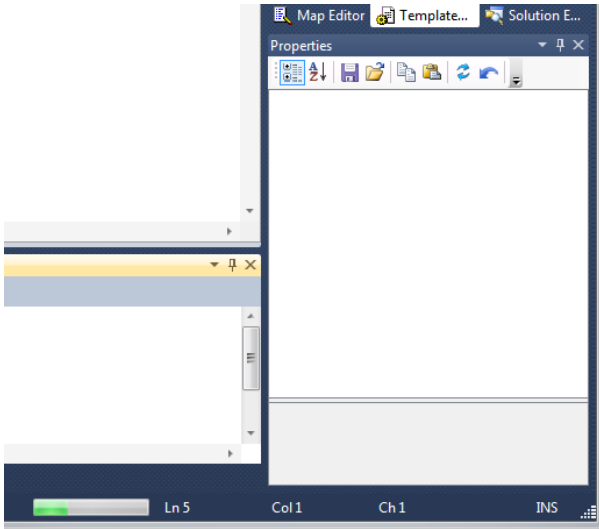
- **AddTextWriter** - Add an [additional output destination](#)
- **Indent** - Increase the indentation level of the output
- **Unindent** - Decrease the indentation level of the output
- **Write** - Write to the generated template without appending a new line
- **WriteLine** - Write to the generated template and append a new line

### The Progress Object

The *Progress* object lets you show a progress bar to the template user when CodeSmith Generator is rendering the template. This is useful when a template takes a long time to render, as it provides a visual cue to the user that CodeSmith Generator has not ceased responding. If you're using CodeSmith Explorer, the progress bar is displayed to the left of the Generate button:



If you're using Visual Studio, the progress bar is displayed in the status bar, to the left of the line and column indicators:



To use the *Progress* object, you'll typically set the maximum value and the amount of change represented by one progress step:

```

this.Progress.MaximumValue = 25;
this.Progress.Step = 1;
  
```

Each time you want to increment the progress indicator on screen, you then call the *PerformStep* method:

```

this.Progress.PerformStep();
  
```

### The CodeTemplateInfo Object

The CodeTemplateInfo object (available through the CodeTemplateInfo property of the CodeTemplate object) can be used to retrieve a variety of information about the current template:

| Property        | Returns   |
|-----------------|---|
| CodeBehind      | Gets the full path to the code-behind file for the template (or an empty string if there is no code-behind file). |
| ContentHashCode | Gets the hash code based on the template content and all template dependencies.                                   |
| DateCreated     | Gets the date the template was created.   |
| DateModified    | Gets the date the template was modified.  |
| Description     | Gets the description.   |
| DirectoryName   | Gets the name of the directory the template is located in.  |

|                |                                     |
|----------------|-------------------------------------|
| FileName       | Gets the name of the template file. |
| FullPath       | Gets the full path to the template. |
| Language       | Gets the template language.         |
| TargetLanguage | Gets the target language.           |

Here's a simple example of using the CodeTemplateInfo object:

```
<%@ CodeTemplate Language="VB" TargetLanguage="Text" Description="Demonstrates CodeTemplateInfo."
%>
<% DumpInfo() %>
<script runat="template">
Public Sub DumpInfo()
Response.WriteLine("Template: {0}", Me.CodeTemplateInfo.FileName)
Response.WriteLine("Created: {0}", Me.CodeTemplateInfo.DateCreated)
Response.WriteLine("Description: {0}", Me.CodeTemplateInfo.Description)
Response.WriteLine("Location: {0}", Me.CodeTemplateInfo.FullPath)
Response.WriteLine("Language: {0}", Me.CodeTemplateInfo.Language)
Response.WriteLine("Target Language: {0}", Me.CodeTemplateInfo.TargetLanguage)
End Sub
</script>
```

The output of running this template will be similar to this:

```
Template: CodeTemplateInfo.cst
Created: 1/1/1973 8:54:19 AM
Description: Demonstrates CodeTemplateInfo.
Location: C:\CodeTemplateInfo.cst
Language: VB
Target Language: Text
```

## Advanced Template Syntax

Advanced Template Syntax covers the following sections:

- [Understanding CodeSmith Generator's Code Behind Model](#)
- [Referencing Assemblies](#)
- [Importing Namespaces](#)
- [Including External Files](#)
- [Sharing Common Code](#)
- [Debugging Templates](#)
- [Using Master Templates](#)
- [Writing to Multiple Outputs](#)

## Understanding CodeSmith Generator's Code Behind Model

When you're writing a CodeSmith Generator template, you're dealing with two distinct kinds of code:

1. The code being generated
2. The scripting code that controls the generation process

As far as CodeSmith Generator is concerned, the first of these is just text, and can be any language at all: VB, SQL, Fortran, COBOL, Esperanto...as long as it can be represented by a string of characters, CodeSmith Generator can generate it. This generated code is stored in the CodeSmith Generator templates, and copied at runtime to the output file, or created on the fly by CodeSmith Generator.

The scripting code is both more and less limited than the generated code. It's more limited in that it can only be VB, C#, or JScript code. But it's less limited in that you have two choices about where to store it. You can either mix it in to the template directly, storing it in <script> blocks, or you can store it in separate code behind files. A code behind file is a source code file containing nothing but scripting code that's attached to a template file by use of attributes within the [CodeTemplate directive](#).

For example, here's a template that makes use of a code behind file:

```
<%@ CodeTemplate Src="VBCodeBehind.cst.vb" Inherits="UtilityCodeTemplate" Language="VB"
TargetLanguage="VB" %>
<%@ Property Name="ClassName" Type="System.String" Category="Options" Description="The name of the
generated class." %>
' This class generated by CodeSmith on <%= DateTime.Now.ToLongDateString() %>
<%= GetAccessModifier(Accessibility) %> Class <%= ClassName %>

    Public Sub New()
    End Sub

    ' Write your class here

End Class
```

Note that this template makes use of a function `GetAccessModifier` and a property `Accessibility`, even though neither one of them is defined in the template. That's because they're defined in a separate code-behind file. Here are the contents of the code-behind file (`VBCodeBehind.cst.vb`):

```
Imports System.ComponentModel
Imports CodeSmith.Engine

' This class contains utility functions that can be
' used across many templates

Public Class UtilityCodeTemplate
    Inherits CodeTemplate

    Private _Accessibility As AccessibilityEnum = AccessibilityEnum.Public

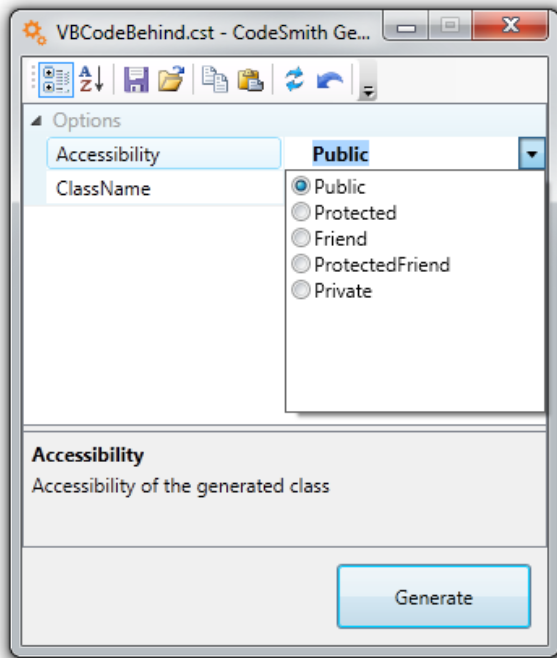
    <Category("Options"), _
    Description("Accessibility of the generated class")> _
    Public Property Accessibility As AccessibilityEnum
        Get
            Return _Accessibility
        End Get
        Set
            _Accessibility = value
        End Set
    End Property

    Public Enum AccessibilityEnum
        [Public]
        [Protected]
        [Friend]
        [ProtectedFriend]
        [Private]
    End Enum

    Public Function GetAccessModifier(ByVal accessibility As AccessibilityEnum) As String
        Select accessibility
            Case AccessibilityEnum.Public
                GetAccessModifier = "Public"
            Case AccessibilityEnum.Protected
                GetAccessModifier = "Protected"
            Case AccessibilityEnum.Friend
                GetAccessModifier = "Friend"
            Case AccessibilityEnum.ProtectedFriend
                GetAccessModifier = "Protected Friend"
            Case AccessibilityEnum.Private
                GetAccessModifier = "Private"
            Case Else
                GetAccessModifier = "Public"
        End Select
    End Function
End Class
```

The `CodeTemplate` directive ties the code-behind file to the template. The `Src` attribute of the directive specifies the filename of the code-behind file, and the `Inherits` attribute of the directive specifies the class in the file that the template is based on. Note that this class must itself inherit, directly or indirectly, from `CodeSmith.Engine.CodeTemplate`.

Because `Accessibility` is defined as a property of the `UtilityCodeTemplate` class, CodeSmith Generator includes it in the template's property sheet when the template is opened in Template Explorer:



There are two main advantages to moving code to a code-behind file:

1. It makes your templates easier to understand by separating the generated code from the scripting code that drives the generation process.
2. It makes it possible to easily reuse utility functions across many templates by moving them to shared code-behind files.

[Click here to download the template and source code file.](#)

## Referencing Assemblies

You can use the `Assembly` directive to reference an external assembly from a template, or to include a source file for dynamic compilation. For example, CodeSmith Generator ships with an assembly named `CodeSmith.CustomProperties.dll` that includes custom editors for file names and string collections. If you'd like to use one of these editors from your own template's property sheet, you need to reference the assembly:

```
<%@ Assembly Name="CodeSmith.CustomProperties" %>
```

[The source code for the CustomProperties assembly is in the Sample folder \(E.G., Documents\CodeSmith Generator\Samples<Version>\Projects\CSharp\CustomPropertiesSample\) of your CodeSmith Generator installation.](#)

## Assembly Directive Attributes

There are two attributes that you can supply to the `Assembly` directive. You must supply one or the other, but not both.

### **Name**

The `Name` attribute specifies the file name of an assembly to reference from the current template. The assembly must exist in the Global Assembly Cache, in the same directory as CodeSmith, in the `CodeSmith\bin` directory, in the `CodeSmith\AddIns` directory, or you can specify a path relative to the template location. If you're working with templates within CodeSmith Generator, the preferred location is the

CodeSmith\AddIns directory.



You can also specify the `Assemblies.FullName` (E.G., `ExampleAssembly, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null`)

### **Src**

The `Src` attribute specifies the relative path to a source file that should be dynamically compiled along with the template.

### **Path**

The `path` attribute is a directory path to the assembly being used.

## **Importing Namespaces**

The `Import` directive is used to import a namespace for use in your template. This lets you refer to types in other assemblies more conveniently. For instance, when you're using the `SchemaExplorer` assembly, you probably don't want to have to prefix every type from that assembly with the name of the assembly. The solution is to include an `Import` directive in your template along with the `Assembly` directive:

```
<%@ Assembly Name="SchemaExplorer" %>  
<%@ Import Namespace="SchemaExplorer" %>
```

### **Import Directive Attributes**

There is one required attribute that you must supply to the `Import` directive.

### **Namespace**

The `Namespace` attribute specifies the fully qualified name of the namespace to be imported.

## **Including External Files**

You can cause CodeSmith Generator to compile the contents of an external file into your template by using an include statement. This can be useful when you have common functions that you want to share between several templates. The include statement takes a single argument which specifies the relative path from the current template to the file to be included:

```
<!-- #include file="CommonScript.cs" -->
```

You can also use an include statement to bring in static template content. It doesn't matter what is in the file that you include; CodeSmith Generator simply inserts the file contents into the template.



There are [other ways](#) to share common code between templates. In most cases, you should use other methods for code-sharing, such as the `Src` attribute on the `CodeTemplate` directive or an `Assembly` directive that imports a source file. That's because these methods require full .NET class files that are easier to edit in other code editors, while an include statement will accept malformed source files.

## **Sharing Common Code**

CodeSmith Generator offers several ways to share common code between templates:

- You can place utility functions in a custom template class and reference it in the `Inherits` attribute of the `CodeTemplate` directive.
- You can base templates on a common template class by using [code-behind files](#).
- You can compile common functions into an assembly, and reference the assembly using an `Assembly` directive.
- You can place common functions into a source code file, and reference the file using an `Assembly` directive.
- You can use [sub-templates](#) to share code between templates.
- You can place common code in a separate source file and use an `include` statement to pull it directly into your `template`.

## **Debugging Templates**

CodeSmith Generator supports debugging by using the CLR's Just-in-Time debugger. This article will show some tips and tricks in setting up CodeSmith Generator templates to use the debugger.

## Allow Debugging in Template

The first step to allow debugging a template is to set the Debug attribute on the `CodeTemplate Declarative` to true.

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Debug="True" %>
```

## Setting a Break Point

In order to get the Just-in-Time debugger to load and stop at a point in your code, you need to use a `System.Diagnostics.Debugger.Break()` statement. If you are using a Code Behind, please remember to import the `System.Diagnostics` namespace.

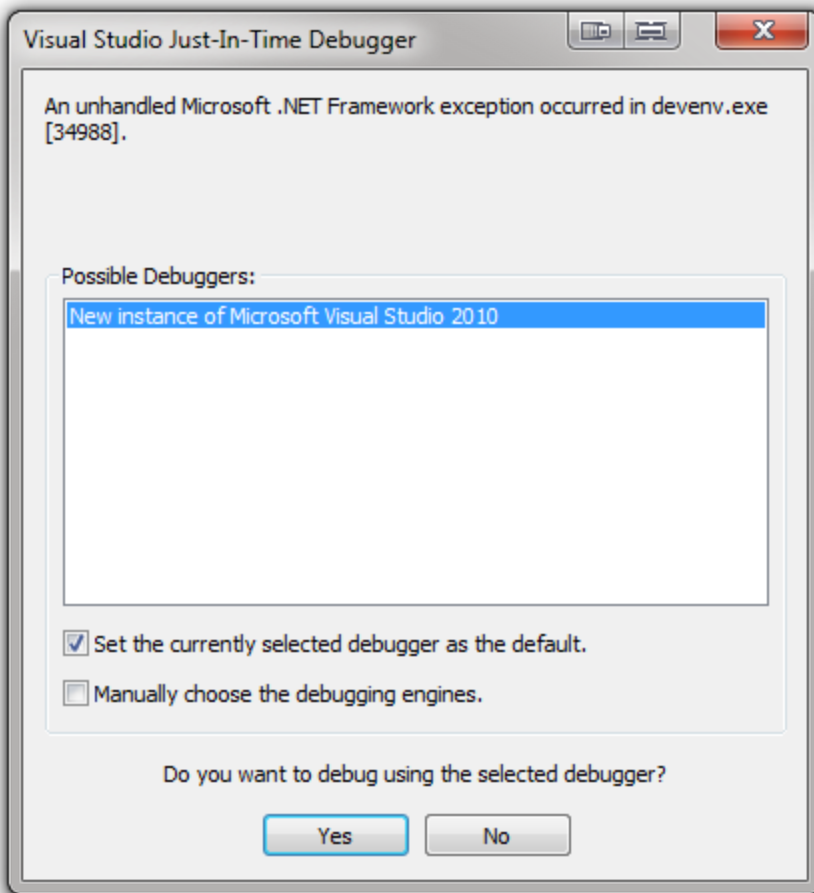
```
System.Diagnostics.Debugger.Launch();  
System.Diagnostics.Debugger.Break();
```



You must call `System.Diagnostics.Debugger.Launch();` before your first `System.Diagnostics.Debugger.Break()` Statement or the process will crash.

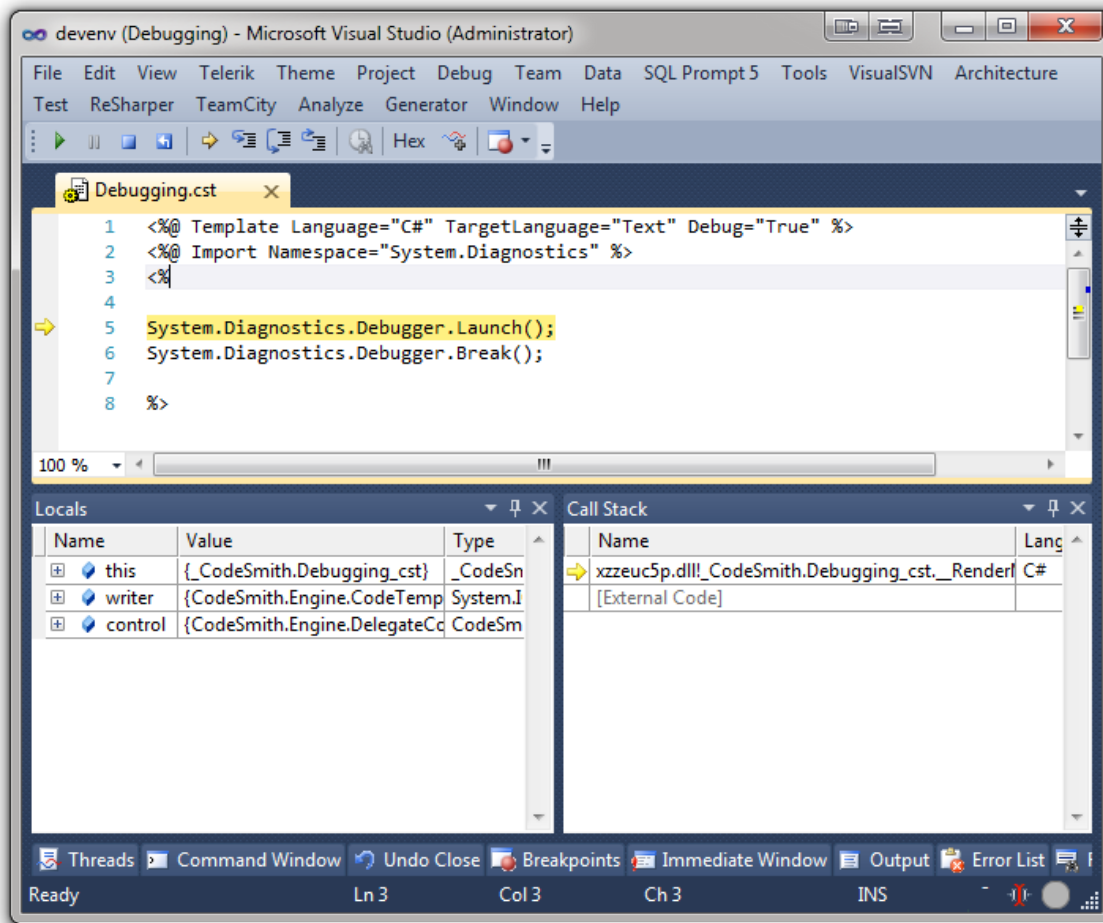
## Using the Debugger

When you execute a template and it encounters a break point, you will see the following dialog.



When you see this dialog choose the debugger you want to use and click on the Yes button.






You can now debug a template just how you would debug any .NET project.

## Debugging in Windows Vista or Windows 7

There are some extra steps that need to be completed before using the Just-In-Time debugger in Windows Vista or Windows 7.

First you need to make sure you have all the latest service packs installed. Next, the debugger in Vista will cause CodeSmith Generator to hang when you finish debugging. You can work around this issue by updating the Just-In-Time debugger setting **DbgJITDebugLaunchSetting**. The setting is found in the registry at [HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\.NETFramework]. Change the value of **DbgJITDebugLaunchSetting** to 2. If you are using a 64bit operating system then you must also set the same key (**DbgJITDebugLaunchSetting**) in this folder [HKEY\_LOCAL\_MACHINE\SOFTWAREWow6432Node\Microsoft\.NETFramework] to 2. This will cause the debugger dialog to be displayed immediately when your code hits a breakpoint. This will also allow control to return to CodeSmith Generator when you continue the execution of the template from the debugger.

 If you are running into issues when trying to debug, run CodeSmith Generator and Visual Studio as an Administrator.

## Troubleshooting

- If you are having trouble with the debugger, try using the CLR debugger as that tends to work better.
- If you are getting the message, "There is no source code available for the current location.", you need to change the default editor for .cst files in Visual Studio to be the "Source Code (Text) Editor".
- If you are having further issues in Vista, make sure to run CodeSmith Generator with full administrator rights by right clicking and choose run as administrator.

## Outputting Trace and Debug Information

One useful way to gather debugging information when a template is not behaving as you expect is to use the methods of the .NET [System.Diagnostics.Trace](#) and [System.Diagnostics.Debug](#) objects. These objects let your code interact with the Debug pane of the [Output window](#). The two objects have exactly the same members; the only difference between the two is that the [System.Diagnostics.Trace](#) object is active at all times, while the [System.Diagnostics.Debug](#) object is only active when you compile your code in [debug mode](#).

This table summarizes some of the useful members of the Trace and Debug objects:

| Member      | Type     | Description   |
|-------------|----------|---|
| Assert      | Method   | Checks for a condition, and displays a message if the condition is false. |
| Fail        | Method   | Displays an error message   |
| Indent      | Method   | Increases the current IndentLevel by one.                                 |
| IndentLevel | Property | Specifies the indent level.   |
| IndentSize  | Property | Specifies the number of spaces in an indent.                              |
| Unindent    | Method   | Decreases the current IndentLevel by one.                                 |
| Write       | Method   | Writes the given information  |
| WriteIf     | Method   | Writes the given information only if a condition is true                  |
| WriteLine   | Method   | Same as Write but appends a new line character after the information.     |
| WriteLineIf | Method   | Same as WriteIf but appends a new line character after the information.   |

## Viewing the Compiled Template Source Code


When you generate or compile a template, CodeSmith Generator creates a compiled assembly. Sometimes, when debugging a template's output, it may be very useful to see the source code of what was compiled.

### Viewing the Compiled Template Source Code

You can view the compiled source code of the current template by right clicking on a template in the Solution Explorer and selecting properties. Next, set the CustomTool property value to **TemplateSourceGenerator**.



Now, when you look at your template in Solution Explorer, you will see a [DependentUpon](#) file ending with .g.cs or .g.vb.

 If a CSharp template is named Template.cst, the file will be called Template.g.cs

Finally, click on this generated document (E.G., Template.g.cs) to view the compiled template source code.

## Using Master Templates

Sub-templates provide a way for you to organize complex code generation processes from a master template. Just as subroutines in a computer program let you call bits of logic from a main program flow, sub-templates let you call bits of code generation logic from a master template.

To use a sub-template, you must first [register the sub-template](#) in the parent template. You can then [merge properties from the sub-template](#) into the parent template, [copy properties from the sub-template](#) to the sub-template, [set properties](#) in the sub-template, and [render the sub-template](#).

Here's a video tutorial on Master Templates!

## Registering Sub-Templates

To register a sub-template, you include a Register directive in the master template. You can include as many Register directives as you like, so one master template can include multiple sub-templates. Sub-templates can be nested.

```
<%@ Register Name="Header" Template="Header.cst" MergeProperties="True"
ExcludeProperties="IncludeMeta" %>
```

### Register Directive Attributes

There are four attributes that you can supply to the Register directive. The Name and Template parameters are required; the others are optional.

#### Name

The Name attribute specifies the type name for the sub-template in the master template. It can be used to create an instance of the sub-template.

#### Template

The Template attribute specifies the relative path to the sub-template.

#### MergeProperties

The MergeProperties attribute specifies whether the properties of the sub-template should be dynamically added to the master template's properties. If you omit this attribute, it defaults to False.

#### ExcludeProperties

The ExcludeProperties attribute specifies a comma-delimited list of properties to be excluded from merging to the master template's property list. You may use \* as a wildcard in the property list.

## Merging Properties into the Parent Template

To merge the properties of a sub-template with a master template, include the MergeProperties="True" attribute in the Register directive for the sub-template. When you do this, the properties of the sub-template will be displayed on the property sheet of the main template when the main template is open in CodeSmith Explorer. This makes it easy to prompt for all the properties that are required for the entire code-generation process on a single property sheet.

```
<%@ Register Name="SubTemplate" Template="SubTemplate.cst" MergeProperties="True" %>
```

## Copying Properties from the Parent Template

You may want to share properties between a master template and sub-templates. For example, suppose you are working with a set of database-oriented templates, and each template defines a string property named Server. When you prompt for this property in the master template, only the master template's copy of the property receives a value.

To set the property in the sub-template, you use the CopyPropertiesTo method of the master template. This method matches properties from the master template to the sub-template on the basis of name and type. If it finds an exact match, it copies the value from the master template to the sub-template. This code snippet shows how you can use this method:

```
// instantiate the sub-template
Header header = this.Create<Header>();

// copy all properties with matching name and type to the sub-template instance
this.CopyPropertiesTo(header);
```

## Setting Properties in a Sub-Template

You can set properties in a sub-template from the main template easily, because they're all available as properties of the instantiated sub-template object. Here's an example:

```
// instantiate the sub-template
Header header = this.Create<Header>();

// include the meta tag
header.IncludeMeta = true;
```

In this case, IncludeMeta is a boolean property defined with a [Property](#) directive in the sub-template.

## Rendering a Sub-Template

After you've registered a sub-template and set its properties, you can render the sub-template. There are several ways to do this. The first is to render the sub-template directly to the output of the main template:

```
// instantiate the sub-template.
Header header = this.Create<Header>();
// render the sub-template to the current output stream.
header.Render(this.Response);
```

Alternatively, you can render the sub-template to a separate file. This is useful when you want to create multiple output files as part of a single code-generation process.

```
// instantiate the sub-template.
Header header = this.Create<Header>();
// render the sub-template to a separate file.
header.RenderToFile("Somefile.txt");
```

The RenderToFile method has several overloads that allow for greater control when rendering content. The overload shown below will prevent the a generated file from overwriting an already existing file called Somefile.txt.

```
// instantiate the sub-template.
Header header = this.Create<Header>();
// render the sub-template to a separate file.
// NOTE: If the file exists then an exception will be thrown.
header.RenderToFile("Somefile.txt", false);
```

The other overloads allow you to use a [Merge Strategies](#) to control how the content should be merged with existing content. Also you can one of the overloads that takes a string file path or OutputFile to specify the the file that the output is [Dependent Upon](#).

## A Sub-Template Example

Here's a simple example so you can see how the various sub-template pieces fit together. This example generates an HTML file from two templates. First, there's a sub-template that generates an HTML header:

```

<%@ CodeTemplate Language="C#" TargetLanguage="HTML" %>
<%@ Property Name="Title" Type="System.String" Optional="False" Category="Options"
Description="Page title." %>
<%@ Property Name="CharSet" Type="System.String" Optional="False" Default="windows-1252"
Category="Options" Description="Character set for the page." %>
<%@ Property Name="IncludeMeta" Type="System.Boolean" Default="True" Optional="False"
Category="Options" Description="Include meta tags." %>
<html>
<head>
<% if (IncludeMeta) { %>
<meta http-equiv="Content-Type" content="text/html; charset=<%= CharSet %>">
<% } %>
<title><%= Title %></title>
</head>

```

Next, the main template generates the body of the HTML file. Note that it uses the sub-template to generate the header:

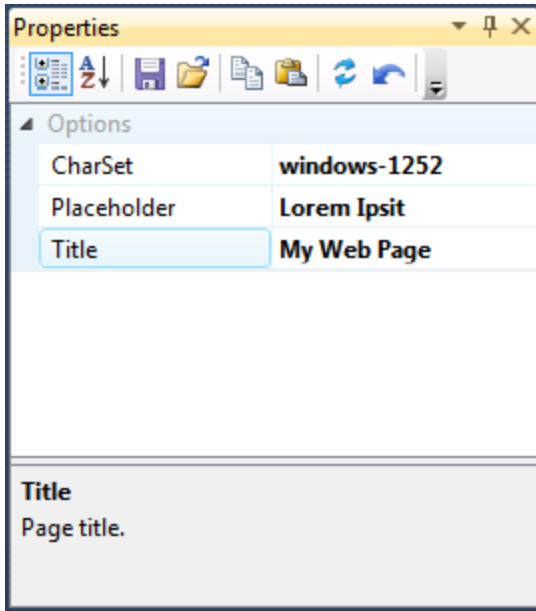
```

<%@ CodeTemplate Language="C#" TargetLanguage="HTML" %>
<%@ Property Name="Title" Type="System.String" Optional="False" Category="Options"
Description="Page title." %>
<%@ Property Name="Placeholder" Type="System.String" Optional="True" Category="Options"
Description="Main placeholder text." %>
<%@ Register Name="Header" Template="Header.cst" MergeProperties="True"
ExcludeProperties="IncludeMeta" %>
<% OutputHeader(); %>
<body>
<h1><%= Title %></h1>
<p><%= Placeholder %></p>
</body>
</html>
<script runat="template">
public void OutputHeader()
{
    Header header = this.Create<Header>();
    // include the meta tag
    header.IncludeMeta = true;
    // copy all properties with matching name and type to the sub-template instance
    this.CopyPropertiesTo(header);
    // render the sub-template to the current output stream
    header.Render(this.Response);
}
</script>

```

When you open the master template, the property sheet shows the Title and Placeholder properties defined in the master template, as well as the CharSet property defined in the sub-template (because of the MergeProperties attribute), but not the IncludeMeta property (because of the ExcludeProperties attribute):

The template's output seamlessly merges the output of the sub-template and the output of the main template:



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>My Web Page</title>
</head>
<body>
<h1>My Web Page</h1>
<p>Lorem Ipsit</p>
</body>
</html>
```

## Writing to Multiple Outputs

CodeSmith Generator lets you send the same output to multiple destinations at one time. To do this, you use the `AddTextWriter` method of the CodeSmith Generator Response object. This method lets you add additional `TextWriter` objects (or objects of any class derived from `TextWriter`) to the list that CodeSmith Generator renders its output to. For example, here's a template that outputs some text to two files at the same time, as well as to CodeSmith Generator's default output window:

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="AddTextWriter Demonstration." %>
<%@ Import Namespace="System.IO" %>
//This template demonstrates using the AddTextWriter method
//to output the template results to multiple locations concurrently.
<script runat="template">
public override void Render(TextWriter writer)
{
    StreamWriter fileWriter1 = new StreamWriter(@"C:\test1.txt", true);
    this.Response.AddTextWriter(fileWriter1);

    StreamWriter fileWriter2 = new StreamWriter(@"C:\test2.txt", true);
    this.Response.AddTextWriter(fileWriter2);

    base.Render(writer);

    fileWriter1.Close();
    fileWriter2.Close();
}
</script>
```

This technique is quite general. You could have a `TextWriter` that streams to a socket or to the Windows clipboard or to file or to a database or to your source code repository or to any other destination you like.



This technique is useful for generating multiple identical copies of the same file. When you need to generate multiple *different* files as part of a single code-generation process, you should use one [sub-template](#) for each file. Call the sub-templates from a master template and use the [RenderToFile](#) method to output each sub-template.

## Driving Templates with Metadata

One of the key features of CodeSmith Generator is that you can use many types of metadata in your templates. Template metadata provides the means for users to interact with templates and customize the output of those templates. You have many choices when defining the metadata in a template:

- You can use [any .NET type](#)
- You can use CodeSmith Generator's [SchemaExplorer](#) to interact with a database
- You can use the [XML support](#)
- You can build your own [custom metadata sources](#), complete with designer and property set support

## Using .NET Types

The easiest way to define metadata is to use one of the scalar .NET types such as System.String or System.Boolean. To define a property using a .NET type, you use a [Property directive](#). CodeSmith Generator automatically allows editing such scalar types directly in its property sheet when the user executes a template.

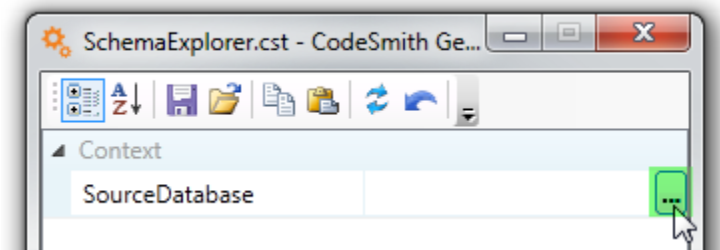
Advanced: [Using Extended Properties to Define Custom Metadata](#)

## Using SchemaExplorer

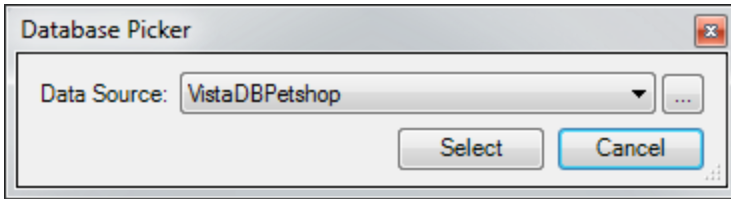
SchemaExplorer is CodeSmith Generator's built-in interface for working with metadata from databases. You can use the classes in SchemaExplorer either programmatically or interactively; often, you'll combine the two approaches. For example, you may want to allow the user to interactively select a database, and then programmatically build a list of all of the tables in the selected database. Here's a template that demonstrates using SchemaExplorer for this purpose:

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="List all database tables" %>
<%@ Property Name="SourceDatabase" Type="SchemaExplorer.DatabaseSchema" Category="Context"
Description="Database containing the tables." %>
<%@ Assembly Name="SchemaExplorer" %>
<%@ Import Namespace="SchemaExplorer" %>
Tables in database "<%= SourceDatabase %>":
<% for (int i = 0; i < SourceDatabase.Tables.Count; i++) { %>
    <%= SourceDatabase.Tables[i].Name %>
<% } %>
```

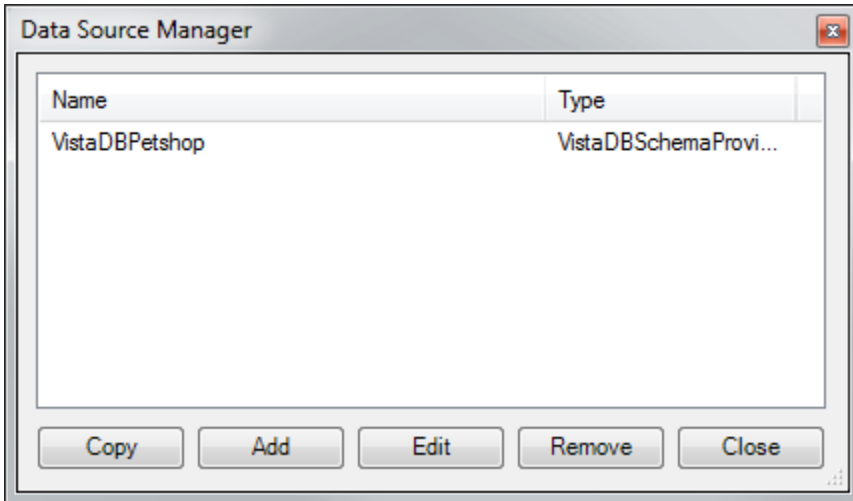
Before you can execute this template, you must supply a value for the SourceDatabase property. When you place your cursor in the property sheet row for this property, CodeSmith Generator will display a builder button (highlighted in green), indicating that there is an external editor hooked up for this property. CodeSmith Generator automatically uses editors built into SchemaExplorer:



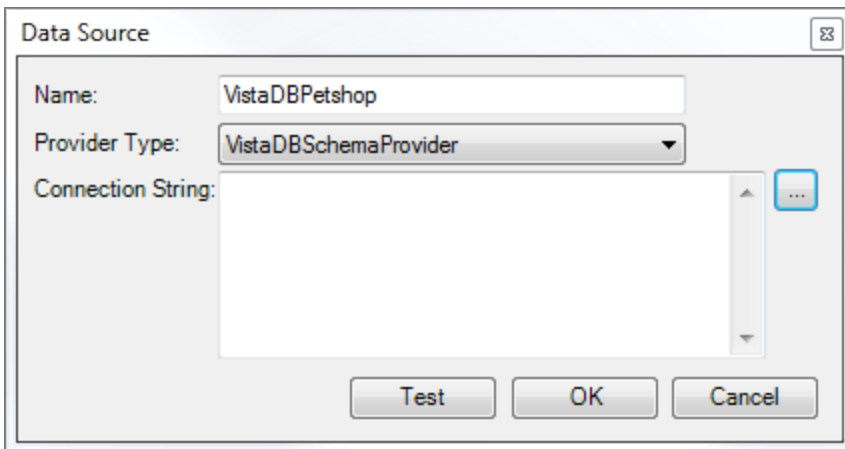
Clicking the builder button opens the Database Picker dialog box. A dropdown list lets you choose from all of the data sources that you have previously defined on your computer. There's also a builder button to define a new data source:



If you click the builder button, SchemaExplorer opens the Data Source Manager dialog box. Here you can see the type of each existing data source, and manage your data sources. You can copy, add, edit, or remove data sources from this dialog box.



If you choose to add a new data source, SchemaExplorer opens the Data Source dialog box. To add a new data source, you must provide a name for the new data source, then select a provider type and type in a connection string.



It's worth noting that CodeSmith Generator ships with many database providers including the following:

- ADOXSchemaProvider
- ISeriesSchemaProvider
- MySQLSchemaProvider
- OracleSchemaProvider
- PostgreSQLSchemaProvider
- SQLAnywhereSchemaProvider
- SqlCompactSchemaProvider
- SQLiteSchemaProvider
- SqlSchemaProvider
- VistaDBSchemaProvider

Here's a sample of the output for this template when it's used with the SQL Server Northwind sample database:



```
Tables in database "Northwind":
Orders
ComponentTypes
Products
Order Details
CustomerCustomerDemo
CustomerDemographics
Region
Territories
EmployeeTerritories
Employees
Categories
Customers
Shippers
Suppliers
```

After the user specifies the `SourceDatabase`, CodeSmith Generator is able to use it as the root of an object model of the entire database.

Check out this video for more information:

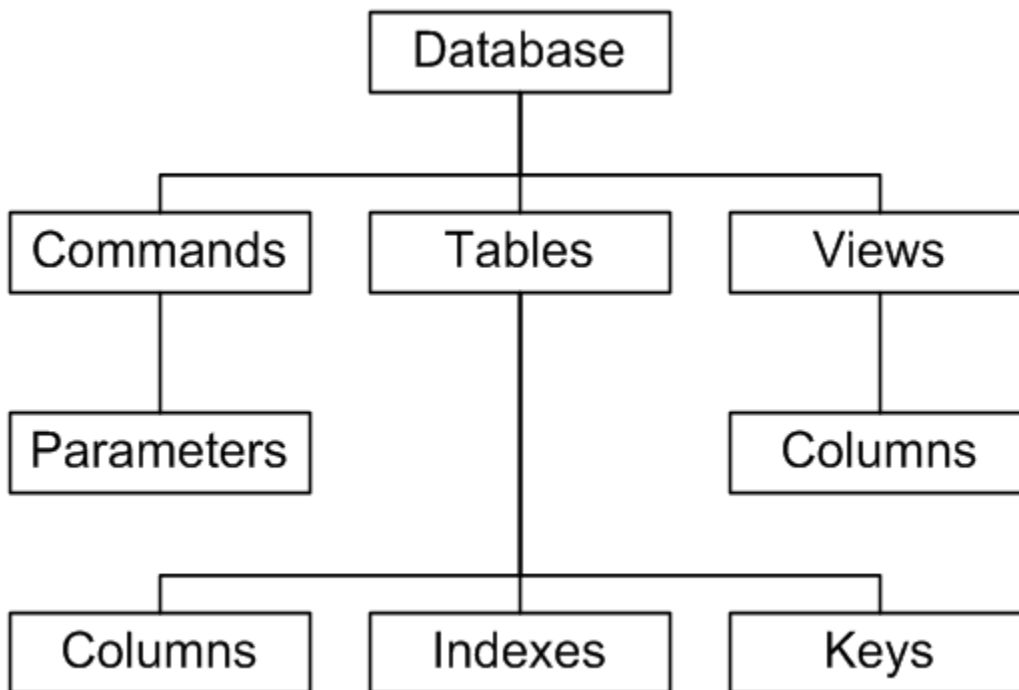


Refer to the [CodeSmith Generator API Reference](#) for a complete listing of the classes and members within the `SchemaExplorer` library.

Advanced: [Using Extended Properties to Define Custom Metadata](#)

## The SchemaExplorer Object Model

Starting with a `DatabaseSchema` object, you can drill down into an object model to obtain further information about the contents of a database selected by the user. This diagram shows the major components of the `SchemaExplorer` object model.



`SchemaExplorer` provides a rich set of collections, objects, and properties that correspond to this object model. For example, the `DatabaseSchema` object exposes a `Commands` property, through which you can retrieve a `CommandSchemaCollection` object. The `Item`

property of this object gives you access to individual `CommandSchema` objects, each of which corresponds to a single command in the database. The properties of the `Command` object allow you to explore the details of the command, retrieving the metadata that you might need to build code based on the command.

## Connection Strings

Valid connection strings in Schema Explorer depend on the provider you're using for a given connection:

If you're using the `SqlSchemaProvider`, connection strings follow the format used by the .NET `SqlConnection.ConnectionString` property.

If you're using the `ADOXSchemaProvider`, connection strings follow the format used by the ADO `ConnectionString` property.

If you're using the `OracleSchemaProvider`, please be sure to check out the link below for creating a connection string as well as the following [how-to article](#).

If you're using the `PostgreSchemaProvider`, please make sure the following statement is included in your `ConnectionString`: **Preload Reader = true;**

For all `SchemaProviders`, please be sure to use the proper `ConnectionString` for the Database Provider you are using. A great resource for building database `ConnectionStrings` can be found [here](#).

## Choosing Objects

`SchemaExplorer` implements designers for four other database object types. These are useful when you need to let users select a particular object or set of objects within a database as part of your template metadata.

`TableSchema` and `TableSchemaCollection`

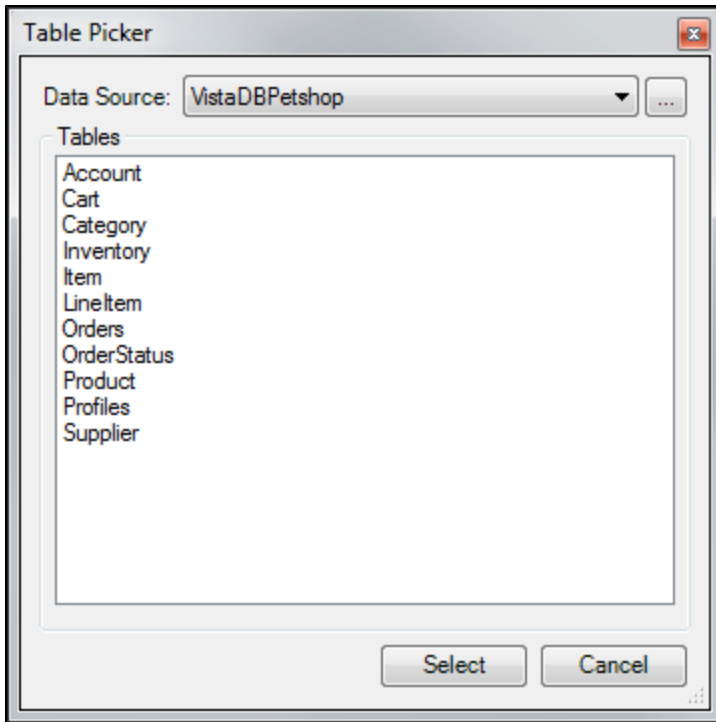
The `TableSchema` editor allows selecting a single table:

```
<%@ Property Name="SourceTable" Type="SchemaExplorer.TableSchema" Category="Database"
Description="Select a table." %>
```

The `TableSchemaCollection` editor allows selecting a group of tables:

```
<%@ Property Name="SourceTables" Type="SchemaExplorer.TableSchemaCollection" Category="Database"
Description="Select a set of tables." %>
```

A property using either of these types will display the Table Picker when the user clicks the Build button in the Properties window. If the property uses the `TableSchema` class, the user can select a single object. If the property uses the `TableSchemaCollection` class, the user can use `Ctrl+click` and `Shift+click` to select multiple objects.



#### ViewSchema and ViewSchemaCollection

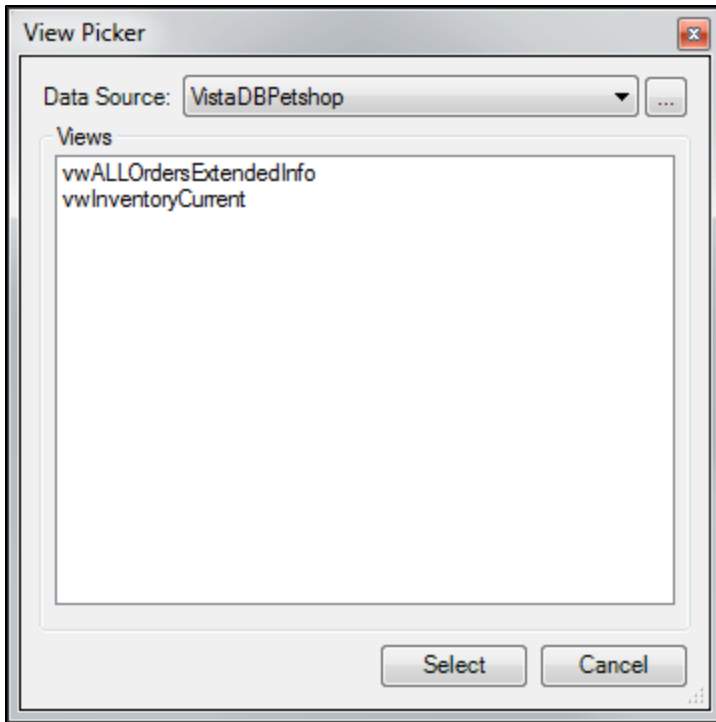
The ViewSchema editor allows selecting a single view:

```
<%@ Property Name="SourceView" Type="SchemaExplorer.ViewSchema" Category="Database"
Description="Select a view." %>
```

The ViewSchemaCollection editor allows selecting a group of tables:

```
<%@ Property Name="SourceViews" Type="SchemaExplorer.ViewSchemaCollection" Category="Database"
Description="Select a set of views." %>
```

A property using either of these types will display the `ViewPicker` when the user clicks the `Build` button in the `Properties` window. If the property uses the `ViewSchema` class, the user can select a single object. If the property uses the `ViewSchemaCollection` class, the user can use `Ctrl+click` and `Shift+click` to select multiple objects.



#### CommandSchema and CommandSchemaCollection

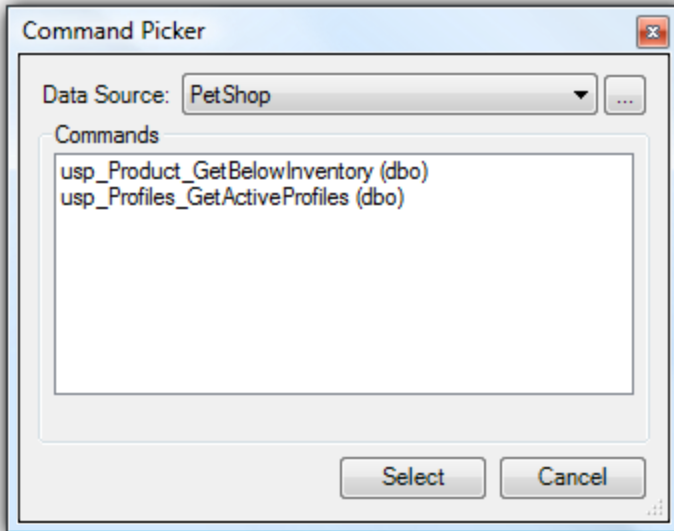
The CommandSchema editor allows selecting a single command:

```
<%@ Property Name="SourceCommand" Type="SchemaExplorer.CommandSchema" Category="Database"
Description="Select a command." %>
```

The CommandSchemaCollection editor allows selecting a group of tables:

```
<%@ Property Name="SourceCommands" Type="SchemaExplorer.CommandSchemaCollection"
Category="Database" Description="Select a set of commands." %>
```

A property using either of these types will display the Command Picker when the user clicks the Build button in the Properties window. If the property uses the CommandSchema class, the user can select a single object. If the property uses the CommandSchemaCollection class, the user can use Ctrl+click and Shift+click to select multiple objects.



### ColumnSchema and ColumnSchemaCollection

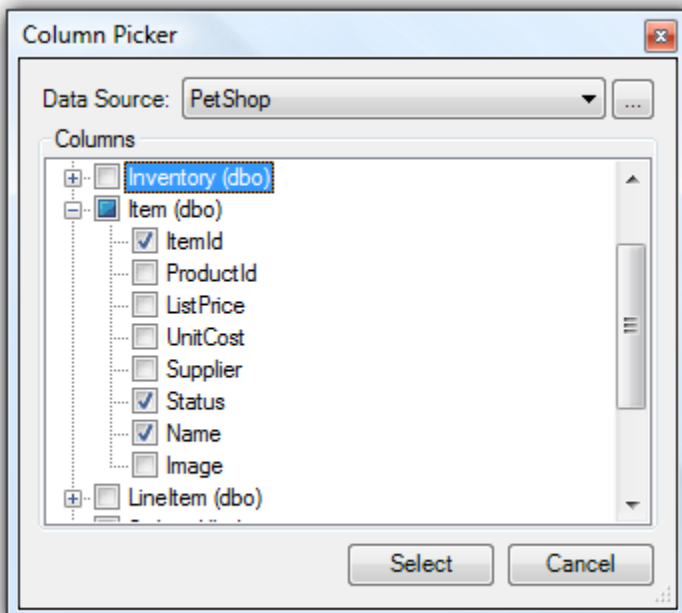
The ColumnSchema editor allows selecting a single command:

```
<%% Property Name="SourceColumn" Type="SchemaExplorer.ColumnSchema" Category="Database"
Description="Select a column." %>
```

The ColumnSchemaCollection editor allows selecting a group of tables:

```
<%% Property Name="SourceColumns" Type="SchemaExplorer.ColumnSchemaCollection" Category="Database"
Description="Select a set of columns." %>
```

A property using either of these types will display the Column Picker when the user clicks the Build button in the Properties window. If the property uses the ColumnSchema class, the user can select a single object. If the property uses the ColumnSchemaCollection class, the user can check the columns to select multiple objects.



## Sorting Collections

SchemaExplorer retrieves the various collections in the order that the database presents them. For all practical purposes, this means that the collections are in a random order. Often, you'll want a collection sorted by name instead. This is easily accomplished by creating a second collection of the same type and using the `Sort` method:

```
TableSchemaCollection tables = new TableSchemaCollection(SourceDatabase.Tables);
tables.Sort(new PropertyComparer("Name"));
```


After running this code, the new `tables` collection will contain all of the tables from the source database, sorted by name.

## Using Extended Properties

SchemaExplorer allows you to retrieve a great deal of information about objects within your database. If you're using a RDBMS (E.G., SQL Server) database, you'll find some of the most useful information in the `ExtendedProperties` collections of the various objects. These collections contain the extended properties that the RDBMS defines for database objects.


### Example

For example, SQL Server defines an extended property that tells you whether a table column is an `identity` column, which you can retrieve with the `"CS_IsIdentity"` extended property key as shown below.

 The `ExtendedProperties` collection is a dictionary defined with a string key and an object value (E.G., `List<string, object>`).

```
Identity Field = <% foreach(ColumnSchema cs in SourceTable.Columns) {
    if( ((bool)cs.ExtendedProperties["CS_IsIdentity"].Value) == true) {
        Response.Write(cs.Name);
    }
} %>
```

A better way to retrieve this value would be to use the `SchemaExplorer.ExtendedPropertyNames` utility class and `ExtendedProperty` Extension methods.


 To use any of the `SchemaExplorer` Extension methods, please be sure to `import` the `SchemaExplorer.Extensions` namespace.

The `SchemaExplorer.ExtendedPropertyNames` class contains string constants of all Schema Provider defined extended properties. This gives you `Intellisense` for extended property key names as well as compile time checking! The below code sample has been updated to use this utility class. The example below will show off how to use `Extended Properties GetByKey` Extension method for retrieving and converting extended property values to the correct type.

```
Identity Field = <% foreach(ColumnSchema cs in SourceTable.Columns) {
    if(cs.ExtendedProperties.GetByKey<bool>(SchemaExplorer.ExtendedPropertyNames.IsIdentity) ==
true) {
        Response.Write(cs.Name);
    }
} %>
```

### Default Extended Properties

CodeSmith Generator defines standard extended properties for table columns, view columns, and command parameters:

 The `ExtendedProperties` collection is a dictionary defined with a string key and an object value (E.G., `List<string, object>`).

#### Table Column

---

| Extended Property Key | SchemaExplorer.ExtendedPropertyName Property Name | Description                           |
|-----------------------|---|---------------------------------------|
| CS_Description        | Description                                       | The Description                       |
| CS_IsRowGuidCol       | IsRowGuidColumn                                   | The Column is a Row Guid              |
| CS_IsIdentity         | IsIdentity  | Identity Column                       |
| CS_IsComputed         | IsComputed  | Computed Column or Index              |
| CS_IsDeterministic    | IsDeterministic                                   | Column is Deterministic               |
| CS_IdentitySeed       | IdentitySeed                                      | Identity Seed                         |
| CS_IdentityIncrement  | IdentityIncrement                                 | Identity Increment                    |
| CS_SystemType         | SystemType  | The System Type (E.G., System.String) |
| CS_Default            | DefaultValue                                      | The default value                     |

#### **View Column**

| Extended Property Key | SchemaExplorer.ExtendedPropertyName Property Name | Description              |
|-----------------------|---|--------------------------|
| CS_Description        | Description                                       | The Description          |
| CS_IsComputed         | IsComputed  | Computed Column or Index |
| CS_IsDeterministic    | IsDeterministic                                   | Column is Deterministic  |

#### **Command Parameter**

| Extended Property Key | SchemaExplorer.ExtendedPropertyName Property Name | Description       |
|-----------------------|---|-------------------|
| CS_Description        | Description                                       | The Description   |
| CS_Default            | DefaultValue                                      | The default value |

In addition, every object has a CS\_Description extended property, but the standard Description property provides a shortcut to the same information.

| Extended Property Key | SchemaExplorer.ExtendedPropertyName Property Name | Description     |
|-----------------------|---|-----------------|
| CS_Description        | Description                                       | The Description |

### **Using CodeSmith to Manage Extended Properties**

CodeSmith Generator offers an easy way to manage Extended Properties through the [Template Explorer's SchemaExplorer Control](#).

#### **Manually Adding Extended Properties**

You can also create your own extended properties within your SQL Server database by using the `sp_addextendedproperty` stored procedure. For example, this T-SQL statement adds a Caption property to the ID column of the Customers table:

```
sp_addextendedproperty 'caption', 'Customer ID', 'user', dbo, 'table', Customers, 'column', id
```

After you execute this statement in your SQL Server database, the Caption property will show up in this column's ExtendedProperties collection in CodeSmith Generator.

## **XML Support**

CodeSmith Generator allows you to store metadata in external XML files. To incorporate XML metadata in your templates, you use an `XmlProperty` directive:

```
<%@ XmlProperty Name="PurchaseOrder" Schema="PO.xsd" Optional="False" Category="Data"
Description="Purchase Order to generate packing list for." %>
```

## XmlProperty Directive Attributes

The XmlProperty directive has six possible attributes. The Name attribute is required, and the other attributes are optional.

### Name

The Name attribute is used as the name of the property when it is displayed on the template's property sheet in CodeSmith Explorer. This is also the variable name that is used to store the value of the property within the template. This must be a legal variable name within the template's language. For example, if the template uses C# as its language, then the name must follow the rules for C# variables. If a schema is specified, the variable will point to a strongly typed object model that CodeSmith generates based on the schema. If no schema is specified, it will point to an instance of the XmlDocument class.

### Schema

The Schema attribute specifies an XSD schema to be used to parse the XML file chosen by the user at runtime.



Specifying a schema file allows CodeSmith to supply IntelliSense help for the XmlProperty instance within your template.



If you do not specify a value for the Schema attribute, then the user can select any XML document at runtime, and the property will return an instance of the XmlDocument class.

### Default

The Default attribute is used to set the default value for this property. If you omit this attribute, then CodeSmith Generator does not supply a default value for the property.

### Category

The Category attribute specifies what category this property should appear under in the CodeSmith Explorer property sheet. If you omit this attribute, CodeSmith Generator will place the property in a category named Misc.

### Description

The Description attribute supplies descriptive text to be displayed at the bottom of the property sheet when this property is selected.

### Optional

The Optional attribute specifies whether or not this property is optional. If a user does not specify a parameter that is not optional then CodeSmith Generator will not let them proceed. A value of true means that a value for the property is not required, and a value of false means that a value for the property is required.

### OnChanged

The OnChanged attribute specifies the event handler to fire when the XmlProperty value changes.

### RootElement

The RootElement attribute specifies the relative or full path to the locate the Root Xml Element.



XmlProperty does not support all variations and features of XSD schemas. In general, if an XSD schema can be successfully loaded into the Visual Studio .NET schema designer then it should work in CodeSmith Generator.

## Additional Information



You can also check out this video on XML Properties for more information!

## XML Property Examples

### XML Property With a Schema

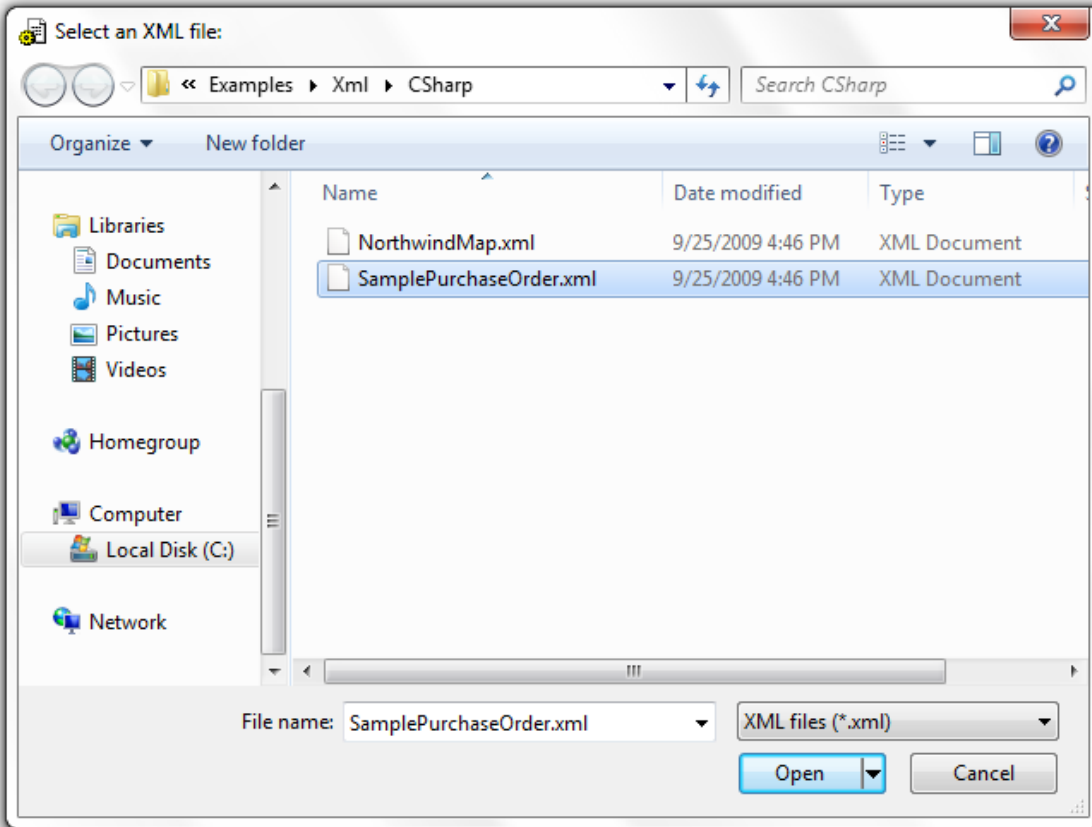
Here's an example of using the `XmlProperty` directive with a schema. Consider first this XSD file, which defines a simple purchase order structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace=http://www.codesmithtools.com/PO
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  xmlns=http://www.codesmithtools.com/PO
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="PurchaseOrder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PONumber" type="xs:string"/>
        <xs:element name="CustomerName" type="xs:string"/>
        <xs:element name="CustomerCity" type="xs:string"/>
        <xs:element name="CustomerState" type="xs:string"/>
        <xs:element name="Items">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Item" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="ItemNumber" type="xs:string" use="required"/>
                  <xs:attribute name="Quantity" type="xs:integer" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Using this schema file, you can define an XML property that accepts purchase order files at runtime for its metadata:

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Description="Create packing list from XML PO."
%>
<%@ XmlProperty Name="PurchaseOrder" Schema="PO.xsd" Optional="False" Category="Data"
Description="Purchase Order to generate packing list for." %>
Packing List
ref: PO#<%= PurchaseOrder.PONumber %>
Ship To:
<%= PurchaseOrder.CustomerName %>
<%= PurchaseOrder.CustomerCity %>, <%= PurchaseOrder.CustomerState %>
Contents:
<% for (int i = 0; i < PurchaseOrder.Items.Count; i++) { %>
<%= PurchaseOrder.Items[i].ItemNumber %>, Quantity <%= PurchaseOrder.Items[i].Quantity %>
<% } %>
```

At run time, the `PurchaseOrder` property will display a builder button in the CodeSmith Generator user interface. Clicking this button opens a file open dialog box which allows the user to browse for an appropriate XML file to use as a metadata source:



Selecting an appropriate XML file generates the packing list. For example, the user might choose this XML file:

```

<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder xmlns=http://www.codesmithtools.com/PO
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <PONumber>5271</PONumber>
  <CustomerName>John Nelson</CustomerName>
  <CustomerCity>Gamonetta</CustomerCity>
  <CustomerState>MS</CustomerState>
  <Items>
    <Item ItemNumber="HM85" Quantity="12"/>
    <Item ItemNumber="JR82" Quantity="4"/>
    <Item ItemNumber="PR43" Quantity="6"/>
  </Items>
</PurchaseOrder>

```

With this resulting generated packing list:

```

Packing List
ref: PO#5271
Ship To:
John Nelson
Gamonetta, MS
Contents:
HM85, Quantity 12
JR82, Quantity 4
PR43, Quantity 6

```

#### XML Property Without a Schema

Here's an example of using the `XmlProperty` directive without a schema. Here's a template that can accept any XML file at runtime:

```

<%@ CodeTemplate Language="VB" TargetLanguage="Text" Description="List top-level nodes in an XML
file." %>
<%@ XmlProperty Name="TargetFile" Optional="False" Category="Data" Description="XML file to
iterate." %>
<%@ Assembly Name="System.Xml" %>
<%@ Import Namespace="System.Xml" %>
Top-level nodes:
<% Dim currNode as XmlNode
currNode = TargetFile.DocumentElement.FirstChild
Do Until currNode Is Nothing%>
    <%= currNode.InnerXml %>
<% currNode = currNode.NextSibling()
Loop %>

```

This template doesn't define a schema for the `TargetFile` property, so that property is presented to the template as an `XmlDocument` object. Thus, it can be processed with the standard XML DOM methods and properties such as `FirstChild` and `NextSibling`. In this case, the template simply loops through the top-level nodes of the document and copies them to the output. For example, if you choose this document as the `TargetFile`:

```

<?xml version="1.0" encoding="UTF-8"?>
<Books>
    <Book>UML 2.0 In a Nutshell</Book>
    <Book>The Best Software Writing</Book>
    <Book>Coder to Developer</Book>
    <Book>Code Complete</Book>
</Books>

```

Then the template's output looks like this:

```

Top-level nodes:
UML 2.0 In a Nutshell
The Best Software Writing
Coder to Developer
Code Complete

```

## Custom Metadata Sources

When you need something beyond the built-in support for .NET types, SchemaExplorer, and XML properties, it's time to explore custom metadata sources. CodeSmith Generator lets you hook just about anything you like to the property grid to use as a metadata source. You'll need to do some coding to enable your custom metadata sources to work smoothly with the rest of CodeSmith Generator, though. The two tasks you may need to accomplish are:

- Adding designer support
- Adding property set support

### Adding Designer Support

If your custom metadata requires a complex user interface (anything beyond the simple text edit control provided by the property grid) to edit, you'll need to add designer support. Otherwise, there won't be any way for the user to enter values for properties that make use of your metadata type. To add designer support, you'll need to build an editor for your type or use an existing `UITypeEditor`. An editor is simply a class that subclasses the .NET `System.Drawing.Design.UITypeEditor` class.



You can find the source code for several CodeSmith Generator custom designers in your extracted template folder. The default template folder is located in the My Documents folder (`Documents\CodeSmith Generator\Samples\<Version>\Projects\CSharp\CustomPropertiesSample`).

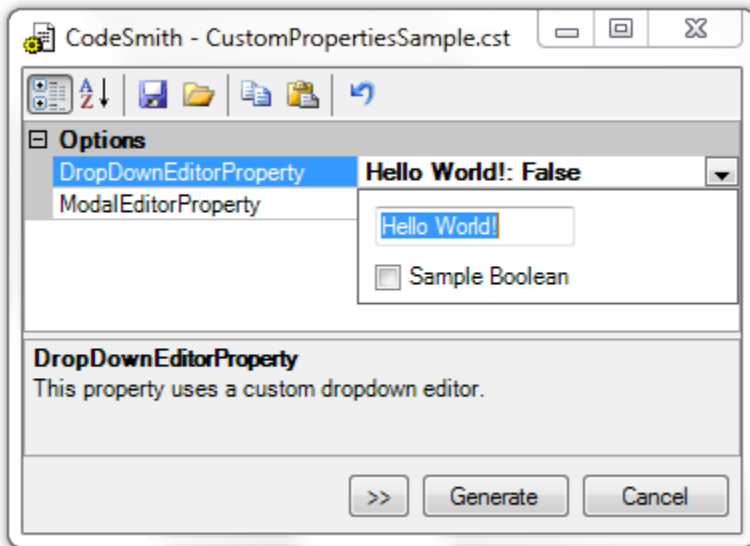
For example, `DropDownEditorProperty` is a class that wraps up a string and a boolean value together into a single piece of metadata. To edit this data, it provides a class, `DropDownEditorPropertyEditor`, which derives from `UITypeEditor`. The declaration of `DropDownEditorProperty` is decorated to indicate that this is the editor class that CodeSmith Generator should use in the property grid:

```
[Editor(typeof(CodeSmith.Samples.DropDownEditorPropertyEditor),
typeof(System.Drawing.Design.UITypeEditor))]
public class DropDownEditorProperty
```

In a template, you can use this metadata type just like any other (although you need to remember to reference its assembly, because CodeSmith Generator doesn't know about this type by default):

```
<@ Property Name="DropDownEditorProperty" Type="CodeSmith.Samples.DropDownEditorProperty"
Category="Options" Description="This property uses a custom dropdown editor." %>
<@ Assembly Name="SampleCustomProperties" %>
```

When the user wants to edit the DropDownEditProperty and clicks in the property sheet, CodeSmith Generator will display the custom designer:




 For more information on building custom designers please read [Building custom UITypeEditors](#).

### Using an predefined UITypeEditor

Here is a compiled list of all UITypeEditors that CodeSmith Generator ships with.

| Name  | Assembly                   | Obsolete   | Description  |
|---|----------------------------|------------|--|
| CodeSmith.Engine.<br><b>CodeFileParserPicker</b>                | CodeSmith.Engine           |            | Allows you to browse for a CSharp or VisualBasic class file.   |
| CodeSmith.Engine.<br><b>XmlPropertyFilePicker</b>               | CodeSmith.Engine           |            | This type editor can be used on a class that supports XML serialization to allow the user to pick an XML file and have that XML file deserialized into the target class. |
| CodeSmith.Engine.<br><b>XmlSchemaFilePicker</b>                 | CodeSmith.Engine           |            | Allows you to browse for an XSD Schema.  |
| CodeSmith.CustomProperties.<br><b>AssemblyFilePicker</b>        | CodeSmith.CustomProperties |            | Allows you to browse for an assembly.  |
| CodeSmith.CustomProperties.<br><b>FileNameEditor</b>            | CodeSmith.CustomProperties |            | Allows you to browse for a file.   |
| CodeSmith.CustomProperties.<br><b>NameValueCollectionEditor</b> | CodeSmith.CustomProperties | <b>YES</b> | Allows you to quickly edit an NameValueCollection.   |


|   |                            |            |  |
|---|----------------------------|------------|--|
| CodeSmith.CustomProperties.<br><b>StringCollectionEditor</b>  | CodeSmith.CustomProperties | <b>YES</b> | Allows you quickly edit a string collection. |
| CodeSmith.CustomProperties.<br><b>XmlSerializedFilePicker</b> | CodeSmith.CustomProperties |            | Allows you to browse for an Xml file.        |

 It is worth noting that the .NET Framework also ships with [many built-in UITypeEditors](#).

## Adding Property Set Support

By default, CodeSmith Generator will try to serialize your custom metadata types automatically using the JSON.NET library. So for most situations, you won't need to do anything special to add CodeSmith Generator Project property serialization support.

If your custom metadata type does not work by default or requires special formatting to save to XML (for instance, it includes information that you want to format in a particular way in the XML file), you'll need to add a property serializer. Otherwise, there won't be any way for the user to save an XML property set file that includes an instance of your metadata type. To add property set support, you'll need to build a serializer for your type. A serializer is simply a class that implements the CodeSmith.IPropertySerializer interface.

 You can find the source code for several CodeSmith Generator custom designers in your extracted template folder. The default template folder is located in the My Documents folder (Documents\CodeSmith Generator\Samples\<Version>\Projects\CSharp\CustomPropertiesSample\

For example, DropDownEditorProperty is a class that wraps up a string and a boolean value together into a single piece of metadata. To serialize this data, it provides a class, DropDownEditorPropertySerializer, which implements IPropertySerializer. The declaration of DropDownEditorProperty is decorated to indicate that this is the serializer class that CodeSmith Generator should use:

```
[PropertySerializer(typeof(CodeSmith.Samples.DropDownEditorPropertySerializer)) | PropertySerializer(type
class DropDownEditorProperty
```

## Generating from Source Code

One of the awesome features of CodeSmith Generator is that you can generate from any kind of metadata. A new feature to CodeSmith Generator is the CodeFileParser which allows you to generate off of existing source code. The CodeSmith.CodeFileParser class can parse any string or file and return an easy to use DOM object.

### Requirements

In order for the CodeFileParser requires that the passed in string or file contents contain valid CSharp or VisualBasic code. The CodeFileParser uses the [public NRefactory libraries](#) under the hood to create the DOM object.

### The CodeFileParser Object

It is very easy to create a new CodeFileParser Instance in code by using the overloaded constructors below. Also, you can use the CodeFileParser by creating a template property. All you need to do is add a new [Property Directive](#) with the type CodeSmith.CodeFileParser to your template.

```
// There are overloads that don't require basePath or parseMethodBodies.
public CodeFileParser(string fileName, string basePath, bool parseMethodBodies)

// There are overloads that don't require parseMethodBodies.
public CodeFileParser(string source, SupportedLanguage language, bool parseMethodBodies)
```

### The Selection Methods

Most of the methods in NRefactory return position information in the form of Location objects, which, while very descriptive, are not the easiest thing to use when trying to take substrings or selections from the existing code.

Because this can be very important when using the object DOM to assist with code generation, we have added several methods to assist with getting substrings and selections; these methods take in Location objects and return strings.

```
public string GetSectionFromStart(Location end)
public string GetSectionToEnd(Location start)
public string GetSection(Location start, Location end)
```

### **The CodeDomCompilationUnit**

To quick and easily walk the DOM, the CodeFileParser exposes a (lazy loaded) property that returns System.CodeDom.CodeCompileUnit object. This is a standard .NET object that contains a complete code graph; this object is the quickest and easiest way to traverse your metadata. For more information about the CodeCompileUnit, please check out [MSDN article](#).

### **The Visitor**

When more advanced or customized information is required, the CodeFileParser exposes the CompilationUnit object, which is capable of taking in a visitor object to traverse the DOM and bring back specific data.

This is an NRefactory feature, and it only requires that your visitor object implement the AbstractAstVisitor class.

### **Example**

We are already using the CodeFileParser in CodeSmith Generator and our Plinqo templates! In CodeSmith Generator we have implemented the CodeFileParser in our [InsertClassMergeStrategy](#); it allows us to parse the existing code file and determine where we need to insert our new content. In [PLINQO for Linq-to-SQL](#) we use the CodeFileParser to assist with our MetaData class merge; it allows us to make a map of all the properties in that class and then preserve their attributes during regeneration.

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text" Debug="False" CompilerVersion="v3.5" %>
<%@ Property Category="2.Class" Name="TheFile" Type="CodeFileParser" Optional="False" %>
<%@ Assembly Name="CodeSmith.CodeParser" %>
<%@ Import Namespace="System.CodeDom" %>

<% foreach(CodeNamespace n in TheFile.CodeDomCompilationUnit.Namespaces) { %>
    Namespace: <%= n.Name %>
    <% foreach(CodeTypeDeclaration t in n.Types) { %>
        Type: <%= t.Name %>
        <% foreach(CodeTypeMember m in t.Members) { %>
            Member: <%= m.Name %>
            <% } %>
        <% } %>
    <% } %>
```

## **Advanced Topics**

Advanced Topics covers the following sections:

- [Upgrading CodeSmith Generator](#)
- [Using the CodeSmith Generator API](#)
- [Auto Executing Generated SQL Scripts](#)
- [Merge Strategies](#)
- [Active vs. Passive Generation](#)
- [Template Caching](#)
- [Building a Custom Schema Provider for SchemaExplorer](#)
- [Using CodeSmith.CustomProperties](#)
- [CodeSmith.BaseTemplates](#)
- [Building a custom UITypeEditor](#)
- [Setting up a DataDirectory for use in Connection Strings](#)
- [Version Control Support](#)

### **Auto Executing Generated SQL Scripts**

If you're generating SQL scripts, it can be useful to execute those scripts right after you've generated them. That way, when you generate scripts that build new objects in a database, you can finish the process by actually building the objects.

The BaseTemplates.ScriptUtility object provides an ExecuteScript method that you can use for this purpose. If you want to execute the script immediately after it's been generated, it's convenient to override the template's OnPostRender method to do so. Here's an example of doing so,

adapted from the StoredProcedures.cst template that ships with CodeSmith Generator:

```
protected override void OnPostRender(string result)
{
    // execute the output on the same database as the source table.
    CodeSmith.BaseTemplates.ScriptResult scriptResult =

    CodeSmith.BaseTemplates.ScriptUtility.ExecuteScript(this.SourceTable.Database.ConnectionString,
        result, new System.Data.SqlClient.SqlInfoMessageEventHandler(cn_InfoMessage));
    Trace.Write(scriptResult.ToString());
    base.OnPostRender(result);
}
```

In this example, SourceTable is a property of type SchemaExplorer.TableSchema. Depending on what metadata you're prompting the user for, you'll need to adjust that part of the code to get a connection to the database where the generated script should be executed.

## Merge Strategies

Merge strategies answer the question: How do I customize generated code without losing my customizations when the code is regenerated? CodeSmith Generator offers you a choice of two different merge strategies:

- [InsertRegion Merge Strategy](#)
- [PreserveRegions Merge Strategy](#)
- [InsertClass Merge Strategy](#)

CodeSmith Generator ships with various Merge Strategy sample templates that can be found in the [Template Explorer](#) under the following folder: \Examples\Merge.



Merge Strategies are supported using the [CodeSmith Console Application](#) and [CodeSmith Project File](#) to generate code.

### Additional Information

### InsertClass Merge Strategy

The InsertClass Merge Strategy is useful when you want to insert your template output into a previously existing class in the output file. At first this may sound very similar to the Insert Region Merge Strategy, and indeed it did start out that way; however, this Merge Strategy has many additional settings and features that separate it from its other fellow Merge Strategies, and that make it a very robust and powerful tool.

### Configuration Options

I think the best way to describe this Merge Strategy is through example; but before we can do that, we must first go over its configuration options.

|                                |                            |  |
|--------------------------------|----------------------------|--|
| <b>Language</b>                | String, Required           | Only Support C# and VB.  |
| <b>ClassName</b>               | String, Required           | Name of the class to insert into.  |
| <b>PreserveClassAttributes</b> | Boolean, defaults to False | Whether or not the merge should preserve the existing classes attributes. By default, the merge tries to replace the entire existing class, which includes the attributes on the top of the class; this option leaves the attributes from the top of the original class. |
| <b>OnlyInsertMatchingClass</b> | Boolean, defaults to False | Insert the whole template output or just the matching class.   |
| <b>MergeImports</b>            | Boolean, defaults to False | Merge the import/using statements of the existing file and generated output.   |

|                       |                        |   |
|-----------------------|------------------------|---|
| <b>NotFoundAction</b> | Enum, defaults to None | <p>What to do if the class is not found in the existing file. There are three options:</p> <p>None: Don't merge anything, just leave the existing file as is.</p> <p>InsertAtBottom: Append the output of the template to the bottom of existing file.</p> <p>InsertInParent. Insert the output of the template at the bottom of a specified parent section (specified by the NotFoundParent property).</p> |
| <b>NotFoundParent</b> | String, no default     | If you specified InsertInParent for the NotFoundAction configuration, you must specify a name for the parent region. This can be the name of a Region or a Class.   |

### Example Configuration...

Language: C#  
 ClassName: "Pet"  
 PreserveClassAttributes: True  
 OnlyInsertMatchingClass: True  
 MergeImports: True

### Existing File

```
using System;
using System.ComponentModel.DataAnnotations;
namespace Petshop
{
    [ScaffoldTable(true)]
    public class Pet
    {
        public int Age { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}
```

### The Generated Output

```
using System;
using System.Text;
namespace Petshop
{
    public class Pet
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string FullName
        {
            get { return String.Format("{0} {1}", FirstName, LastName); }
        }
    }
}
```

### Insert Class Merge Strategy Result!



```

using System;
using System.ComponentModel.DataAnnotations;
using System.Text;
namespace Petshop
{
    [ScaffoldTable(true)]
    public class Pet
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string FullName
        {
            get { return String.Format("{0} {1}", FirstName, LastName); }
        }
    }
}

```

## InsertRegion Merge Strategy

The InsertRegion merge strategy is useful when you need to generate a single region of code within a file that is otherwise not authored by CodeSmith Generator. The file must already exist with the appropriate region marked. CodeSmith Generator preserves the rest of the file untouched. When using the InsertRegion merge strategy, you specify an initialization string in this format:

```
RegionName=<RegionName>;Language=<Language>
```

For example,

```
/merge:InsertRegion="RegionName=Sample Generated Region;Language=C#;"
```

Given this initialization string, CodeSmith Generator will search for a region named "Sample Generated Region" marked by C# style region markers. The generated code will be inserted in place of the contents of this region. The Language attribute in the initialization string is a key into the HKEY\_CURRENT\_USER\Software\CodeSmith\<VERSION>\MergeStrategyAlias registry node. This node contains regular expressions for defining the region markers for each supported language. By default, CodeSmith Generator recognizes region markers for VB, C#, and T-SQL, but you can add your own regular expressions to the file to extend this support if you need to.



If you do not specify a Language attribute in the initialization string, then the TargetLanguage attribute in the template's CodeTemplate directive is used as a key instead.

## Example

Here's an example so you can see how all the pieces fit together. First, a template, Copyright.cst, that can generate boiler plate copyright notices in a format suitable for insertion in Visual Basic code:

```

<%@ CodeTemplate Language="VB" TargetLanguage="VB" Description="Copyright notice generator." %>
<%@ Property Name="CompanyName" Type="System.String" Category="Strings" Description="Your company
name." %>
<%@ Property Name="ClientName" Type="System.String" Category="Strings" Description="Client company
name." %>
' This module is delivered as licensed content as defined
' in the contract between <%= ClientName %> and <%= CompanyName %>.
' Copyright (c) <%= System.DateTime.Now.Year %> <%= CompanyName %>
' All other rights reserved.

```

Next, the Copyright.xml property set XML file with metadata for this template:

```

<?xml version="1.0" encoding="utf-8"?>
<codeSmith>
  <propertySet>
    <property name="CompanyName">CodeSmith Tools, LLC</property>
    <property name="ClientName">Doe Industries</property>
  </propertySet>
</codeSmith>

```

HelloWorld.vb is a Visual Basic source code file with a region suitable for inserting the generated copyright notice. Note that this file also contains some code that CodeSmith Generator should leave untouched:

```

Public Class HelloWorld
#Region "Copyright Notice"
  'CodeSmith will insert the copyright notice here
#End Region
  Public Sub New()
  End Sub
  Public Sub SayHello()
    ' CodeSmith will leave this code intact
    MessageBox.Show("Hello World")
  End Sub
End Class

```

The result is to change HelloWorld.vb to look like this:

```

Public Class HelloWorld
#Region "Copyright Notice"
' This module is delivered as licensed content as defined
' in the contract between Doe Industries and CodeSmith Tools, LLC.
' Copyright (c) 1973 CodeSmith, LLC
' All other rights reserved.
#End Region
  Public Sub New()
  End Sub
  Public Sub SayHello()
    ' CodeSmith will leave this code intact
    MessageBox.Show("Hello World")
  End Sub
End Class

```

CodeSmith Generator preserves everything outside of the specified region, including the region markers. This means that you can change the metadata for the generation process and regenerate as often as you want without affecting anything outside of the specified region.

## PreserveRegions Merge Strategy

The PreserveRegions merge strategy is useful when you need to preserve multiple custom regions in a file that is otherwise authored by CodeSmith Generator. The file must already exist with the custom sections marked by appropriate region markers. CodeSmith

Generator transfers the marked regions to the template output when regenerating the file. When using the PreserveRegions merge strategy, you specify an initialization string in this format:

```
RegionNameRegex=<RegexExpression>;Language=<Language>
```

For example,

```
RegionNameRegex=^[ \t]*(?i:Custom);Language=T-SQL;
```

Given this initialization string, CodeSmith Generator will search for a region named "Sample Generated Region" marked by C# style region markers. The generated code will be inserted in place of the contents of this region. The Language attribute in the initialization string is a key into the HKEY\_CURRENT\_USER\Software\CodeSmith\<VERSION>\MergeStrategyAlias registry node. This node contains regular expressions for defining the region markers for each supported language. By default, CodeSmith Generator recognizes region markers for VB, C#, and T-SQL, but you can add your own regular expressions to the file to extend this support if you need to.



If you do not specify a Language attribute in the initialization string, then the TargetLanguage attribute in the template's CodeTemplate directive is used as a key instead.

Here's an example so you can see how all the pieces fit together. First, a template, CustomClass.cst. Note that the template defines two empty C# regions. These are the regions that will be used to preserve custom code that already exists in the output file.

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Custom class generator." %>
<%@ Property Name="ClassName" Type="System.String" Description="Name of the class." %>
#region Keep copyright
#endregion

class <%= ClassName %>
{
    public <%= ClassName %>()
    {
        // Insert default constructor code here
    }

    #region Keep custom methods
    #endregion
}
```

Next, the existing output file, Engine.cs:

```
#region Keep copyright
Copyright (c) 1973 CodeSmith Tools, LLC
#endregion

class Engine
{
    public Engine()
    {
        // Insert default constructor code here
    }

    #region Keep custom methods
    public string UniqueID()
    {
        return("E8472");
    }
    #endregion
}
```

And the resulting changes to Engine.cs:

```
#region Keep copyright
Copyright (c) 1973 CodeSmith LLC
#endregion

class Driver
{
    public Driver()
    {
        // Insert default constructor code here
    }


    #region Keep custom methods
    public string UniqueID()
    {
        return("E8472");
    }
    #endregion
}
```

## Defining Your Own Merge Strategy

Merge strategies are carried out by classes that implement the [CodeSmith.Engine.IMergeStrategy](#) interface. After defining your own merge strategy, there are two ways that you can use it. First, you can specify a fully-qualified assembly name when calling the merge strategy from the command line:

```
/merge:MyMergeAssembly.MyMergeStrategy="MergeParameters=Sample Merge Parameters;Language=C#;"
```

Also, you can register your merge strategy with CodeSmith Generator. This allows you to refer to your merge strategy by name and not by the types FullName.

 Adding your merge strategy to the alias list allows you to call it by name when you use the /merge switch.

To do this, you will need to add a registry entry to the following node:

```
HKEY_CURRENT_USER\Software\CodeSmith\<VERSION>\MergeStrategyAlias\<NUMBER THAT DOESN'T EXIST (E.G., 5)>
```

Finally, you will need to create the following string registry values: Name and TypeName. Here is an entry of an example that ships with CodeSmith Generator, the contents of the code below is an [registry export](#).

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\CodeSmith\<VERSION>\MergeStrategyAlias\5]
"Name"="PreserveRegions"
"TypeName"="CodeSmith.Engine.PreserveRegionsMergeStrategy,CodeSmith.Engine"
```

## Active vs. Passive Generation

Broadly speaking, there are two different types of code generators: passive code generators and active code generators.

*Passive code generators* generate code once and then give up all responsibility for it. The wizards and builders that you find in modern IDEs are typically passive code generators. They're good for coming up with code that the developer later customizes, but once the code has been generated, a passive code generator can't regenerate it with changes.

In contrast, *active code generators* are designed to maintain a link with the code that is generated over the long term by allowing the generator to be run multiple times over the same code. The key point to keep in mind about active code generators is that the template is the source code.

Suppose you're generating 500 class files from a single template. With an active code generator, if you find a bug in the architecture of those classes (say, you've made a mistake in the way that you're handling object persistence), it's not a huge problem. You just fix the one template and regenerate the 500 classes. This obviously saves you an incredible amount of time over fixing the same bug over and over again in 500 separate class files.

But what happens when a template can't generate everything that needs to appear in the source code file? Suppose some of those 500 classes need custom methods, and the custom methods are different in different classes. For an active code generator to be effective, it must provide some way for a developer to customize its output, and then allow code regeneration without destroying those customizations.

By default, CodeSmith Generator doesn't allow for custom code in the files that it generates. When you execute a template, it overwrites any existing output file completely. But there are ways to use CodeSmith Generator in conjunction with custom code. Here are three strategies to enable active code generation and custom code together with CodeSmith Generator:

- Use inheritance
- Use merge strategies
- Use .NET 2.0 partial classes

## Using Inheritance to Enable Active Generation

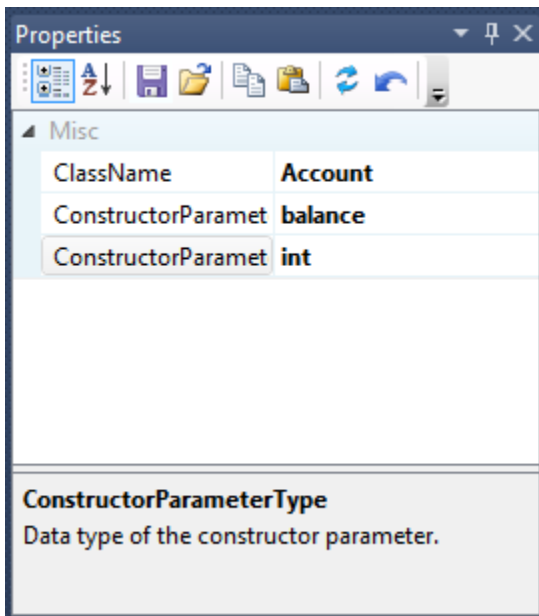
One way to enable active code generation with CodeSmith Generator is to use CodeSmith Generator to generate a base class, and then to customize a derived class. When you regenerate the base class, CodeSmith Generator doesn't touch the code in the derived class.

As a simple example, you might design this template to use in a financial application:

```
<#@ CodeTemplate Language="C#" TargetLanguage="C#" Description="Base class generator." %>
<#@ Property Name="ClassName" Type="System.String" Description="Name of the class." %>
<#@ Property Name="ConstructorParameterName" Type="System.String" Description="Constructor
parameter name." %>
<#@ Property Name="ConstructorParameterType" Type="System.String" Description="Data type of the
constructor parameter." %>
class <%= ClassName %>
{
    <%= ConstructorParameterType %> m_<%= ConstructorParameterName %>;

    public <%= ClassName %>(<%= ConstructorParameterType %> <%= ConstructorParameterName %>)
    {
        m_<%= ConstructorParameterName %> = <%= ConstructorParameterName %>
    }
}
```

You might execute this template with the following metadata:



The resulting generated class looks like this:

class Account

```
{
    int m_balance;

    public Account(int balance)
    {
        m_balance = balance
    }
}
```

With the generated code stored in Account.cs, you could then write a second class by hand and store it in Checking.cs:

class Checking : Account

```
{
    public Checking : base(0)
    {
    }
}
```

Now, suppose that your requirements change and you decide that the account balance should really be a floating-point value. You can change the ConstructorParameterType property to double, regenerate the Account.cs file, and recompile without touching the handwritten code in Account.cs. As long as you don't insert any custom code directly in the base class, you can regenerate that class as often as you like.

## Using Merge Strategies to Enable Active Generation

A second way to enable active code generation with CodeSmith Generator is to use the [CodeSmith Generator Console Application](#) with [merge strategies](#) to generate code. Merge strategies allow you to generate specific portions of a file, while allowing the developer to customize the remainder of the file. Merge strategies provide a flexible way to enable active code generation, provided that developers maintain the discipline to avoid putting custom code inside of the regions that will be overwritten by CodeSmith Generator.

For example, consider this template for generating HTML pages with some standard scaffolding around a user-entered body:

```
<%@ CodeTemplate Language="C#" TargetLanguage="HTML" %>
<%@ Property Name="Title" Type="System.String" Optional="False" Category="Options"
Description="Page title." %>
<%@ Property Name="CharSet" Type="System.String" Optional="False" Default="windows-1252"
Category="Options" Description="Character set for the page." %>
<%@ Property Name="IncludeMeta" Type="System.Boolean" Default="True" Optional="False"
Category="Options" Description="Include meta tags." %>
<%@ Property Name="Copyright" Type="System.String" Optional="False" Default="Copyright (c)
MyCompany.com" Category="Options" Description="Character set for the page." %>
<html>
<head>
<% if (IncludeMeta)
{ %>
<meta http-equiv="Content-Type" content="text/html; charset=<%= CharSet %>">
<% } %>
<title><%= Title %></title>
</head>
<body>
<h1><%= Title %></h1>
<!-- region Keep body -->
<!-- endregion -->
<p><%= Copyright %></p>
</body>
</html>
```

Given this template, you can use the [PreserveRegion Merge Strategy](#) to enable active code generation. As long as the user limits their changes to the "Keep body" region, you can regenerate the meta tag, title, and copyright statement for the page as often as you like without destroying their changes.



To successfully use the PreserveRegion Merge Strategy with this template, you need to define the region markers for the HTML template:

```
<languageRegionDefinitions>
  <languageKeyList>
    <key>HTML</key>
  </languageKeyList>
  <regionStartRegex>[ \t]\<!--#?(?i:region)(?<name>[\r\n])?\r?\n</regionStartRegex>
  <regionEndRegex>^[ \t]\<!--#?(?i:endregion.\r?\n</regionEndRegex>
</languageRegionDefinitions>
```

## Using Partial Classes to Enable Active Generation

.NET 2.0 offers a new feature that enables active code generation scenarios in both C# and Visual Basic .NET: partial classes. With partial classes, the code for a single class can be split across multiple class declarations, in one or more files. At compile time, the compiler locates all of the pieces of the class and assembles them into a single compiled class.

In C#, partial class definitions look like this:

```
partial class Class1
{
    public void Method1
    {
        // code to implement Method1
    }
}

partial class Class1
{
    public void Method2
    // code to implement Method2
}
}
```

In Visual Basic, the same example looks like this:

```
Partial Public Class Class1
    Public Sub Method1
        ' Code to implement Method1
    End Sub
End Class

Partial Public Class Class1
    Public Sub Method2
        ' Code to implement Method2
    End Sub
End Class
```

In either case, you can enable active generation by generating the code for Method1, while keeping the handcrafted code for Method2 in a separate file, untouched by CodeSmith Generator. At compile time, the appropriate compiler will knit the two files together into a single unified class.

## Template Caching

CodeSmith Generator uses a technique called template caching to speed up the process time of generating templates. If the template's content and dependencies have not changed, CodeSmith Generator is able to use the already compiled assembly to render the template's output, rather than recompiling the template.

For CodeSmith Generator to be able to use template caching, the template must be unchanged since the last time it was compiled. To determine

this, CodeSmith Generator checks the source code of the template for changes, and also recurses into any templates and source files referenced through `Assembly` or `Register` directives.

Template caching makes a big difference in the performance of CodeSmith Generator when you're repeatedly executing the same template without modifying the template itself. You'll see a nice performance boost when you just execute a template (the Properties window will open more quickly), but the real benefit comes when you integrate CodeSmith Generator into your build process. With template caching, using CodeSmith Generator through the [batch generation process](#) or [CodeSmith Generator Console Application](#) is substantially faster.

## Version Control Support

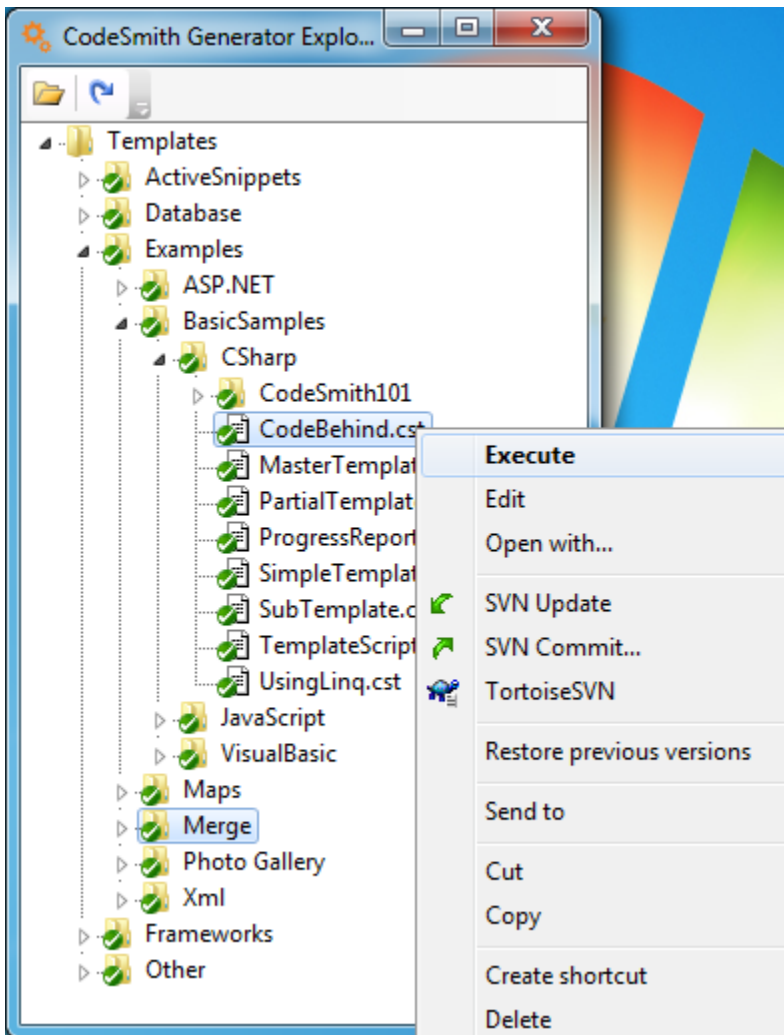
### Exclusive Checkouts

In [CodeSmith Generator 6.0](#), we made some changes that will help users who are using exclusive checkout version control systems like Microsoft Team Foundation Server (TFS), Vault and Visual SourceSafe. In the past, if you generated a bunch of files using CodeSmith Generator, it would overwrite the file contents and change the modified date on files even if the generated content was exactly the same as the existing content. This would cause the version control system to checkout every single file and treat them as if they were changed and needed to be checked in. Most version control systems would then see that the content hadn't actually changed and ignore the check-in request, but it was still a pain when you went to check-in and would see hundreds of modified files in the list.

We also went through a big round of testing to make sure that everything works as expected using the exclusive checkout systems inside of [Visual Studio](#). When you make a change to a [Generator Project file](#), it is automatically checked out as you would expect. We tried to make sure everything works as smoothly as possible.

### Template Explorer

[Template Explorer](#) has rich integration with Windows which also includes all of your Windows Explorer context menus. This gives you the ability to update from source control and much more. The image below shows the ability to update or commit to a SVN repository using the Windows Explorer context menus.






## Building a Custom Schema Provider for SchemaExplorer


If you have a custom data source that looks like a database (that is, it exposes data in tables and columns, perhaps with indexes, views, and commands), you may find it convenient to hook your data source into SchemaExplorer. If you do this, users will be able to use the standard SchemaExplorer [user interface components](#) to retrieve data from your data source, and you can use the [SchemaExplorer object model](#) to work with the data in your templates.


You can integrate your own data with SchemaExplorer by building a custom Schema Provider. In this tutorial we will show you how to build and debug a custom Schema Provider.

 You can find the source code for all of CodeSmith Generators' Schema Providers in your extracted templates folder. The default template folder is located in the My Documents folder. You can find all of the extracted Schema Provider source code in the following directory: Documents\CodeSmith Generator\Samples\<Version>\Projects\CSharp\

### Creating a Custom Schema Provider

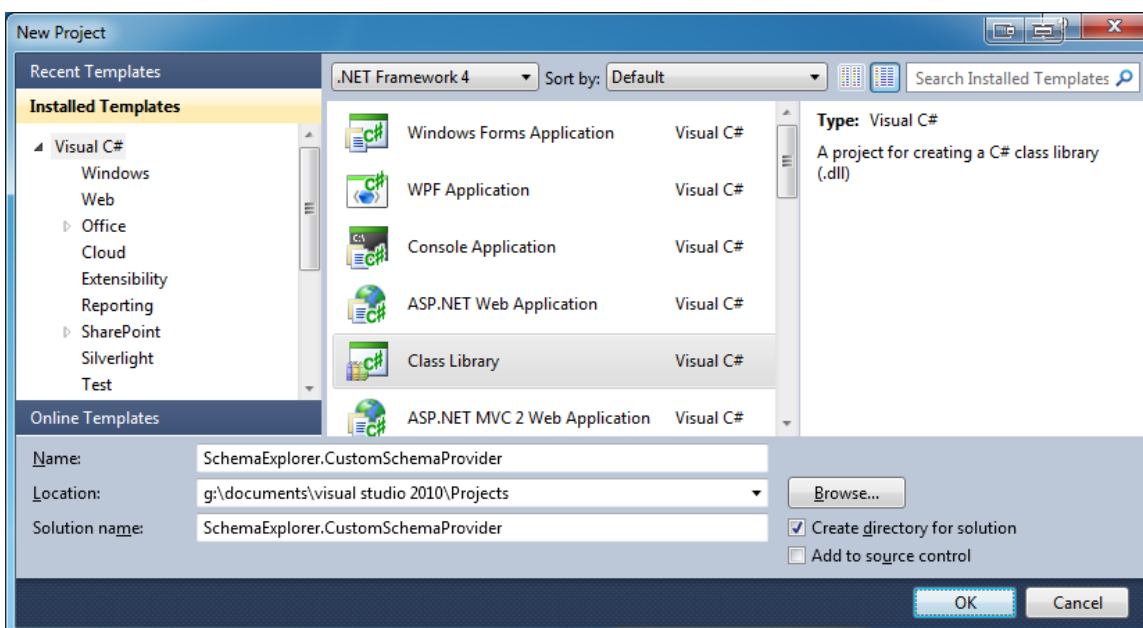
You can integrate your own data with SchemaExplorer by building a custom schema provider. To create your own schema provider simply create a new assembly which includes a public class that implements the **SchemaExplorer.IDbSchemaProvider** interface. Also, the assembly name must end with **SchemaProvider.dll** (for example, SchemaExplorer.Custom**SchemaProvider.dll**). All of the CodeSmith Generator Schema Providers are prefixed with "**SchemaExplorer.**" (E.G., SchemaExplorer.SqlSchemaProvider.dll). It is recommended that you also choose to follow this naming convention.

 It is recommended to follow the naming pattern defined above as it will ensure that an assembly is compiled that follows our naming conventions. Please note that when looking at other provider source code, a default namespace **SchemaExplorer** will be used. **It is not required that your SchemaProvider reside in a namespace named SchemaExplorer.**

 You can find the source code for all of CodeSmith Generators' Schema Providers in your extracted templates folder. The default template folder is located in the My Documents folder. You can find all of the extracted Schema Provider source code in the following directory: Documents\CodeSmith Generator\Samples\<Version>\Projects\CSharp\

### Creating a new Schema Provider

The next step is to create a new Schema Provider by opening up Visual Studio and add a new CSharp or Visual Basic Class Library named **SchemaExplorer.CustomSchemaProvider**. Please note that this can be any name that you choose as long as you follow the naming criteria specified above.

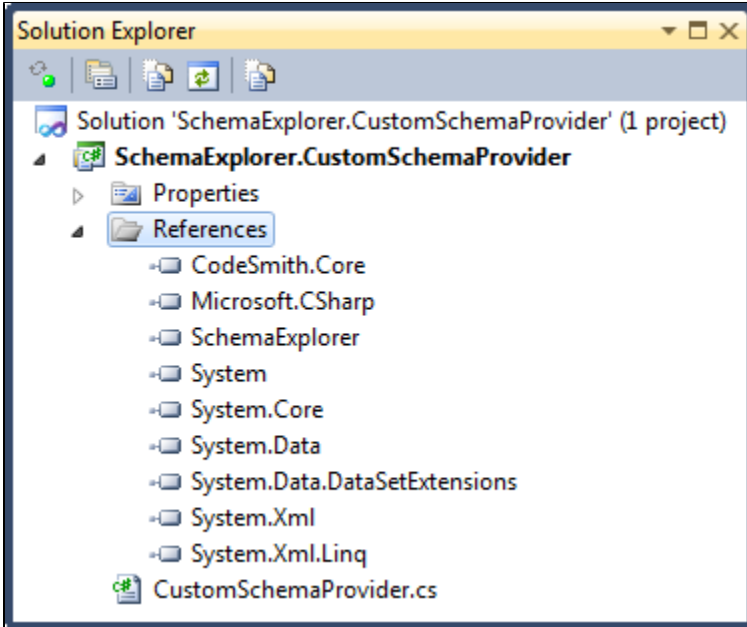


Once the project has been created, we will want to rename the Class1.cs file to CustomSchemaProvider.cs and rename the class name

to CustomSchemaProvider.

### Adding CodeSmith Generator References

We will now add references to our new Class Library project so we can start implementing the SchemaExplorer Interfaces that allows CodeSmith Generator to consume our new Schema Provider. *Note: To add a project reference, please see this guide.* You will need to navigate to the CodeSmith Generator Program Files folder and add a reference to the following two assemblies: **Addins\SchemaExplorer.dll** and **bin\CodeSmith.Core.dll**. The references should now show up in the Visual Studio Solution Explorer Tool Window.



### Inheriting the SchemaExplorer Schema Provider Interfaces

Now it is time to implement the SchemaExplorer Interfaces so CodeSmith Generator can talk to the new provider. The main Interface that is required is called SchemaExplorer.IDbSchemaProvider. This interface implements the core functionality for populating the SchemaExplorer objects (E.G., TableSchema, ColumnSchema, etc...). There is a second interface that you can implement called SchemaExplorer.IDbConnectionStringEditor. When this interface is implemented, it tells CodeSmith Generator that there is an available ConnectionString designer and allows you to show a designer when the user clicks on the designer button.



You can find the source code for all of CodeSmith Generators' Schema Providers in your extracted templates folder. The default template folder is located in the My Documents folder. You can find all of the extracted Schema Provider source code in the following directory: Documents\CodeSmith Generator\Samples\<Version>\Projects\CSharp\

It is highly recommended to take a look at the existing Open Source Schema Provider source code for all of the CodeSmith Generator Schema Providers as a reference when building a new Schema Provider. This can be found in your My Documents folder as described in the tip above. After we have implemented the two interfaces above, our new CustomSchemaProvider class should look like this:

```
using System;

namespace SchemaExplorer.CustomSchemaProvider
{
    public class CustomSchemaProvider : SchemaExplorer.IDbSchemaProvider,
        SchemaExplorer.IDbConnectionStringEditor
    {
    }
}
```

The next step is to implement the interfaces defined above. I told Visual Studio to implement the interfaces to save me from writing a lot of the simple implementation details. Below is what the default implementation looks like after it has been created by Visual Studio.

```
#region Implementation of IDbSchemaProvider
```

```

/// <summary>
/// Gets the name of the schema provider (E.G. SqlSchemaProvider).
/// </summary>
public string Name { get; private set; }

/// <summary>
/// Gets the description for the schema provider (E.G. SQL Server Schema Provider)..
/// </summary>
public string Description { get; private set; }

/// <summary>
/// Gets the name of the database.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <returns>The name of the database</returns>
public string GetDatabaseName(string connectionString)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the extended property collection for a given schema object.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="schemaObject">Any type that derives from SchemaObjectBase. (E.G. DatabaseSchema,
TableSchema, ColumnSchema, ViewSchema, ViewColumnSchema, IndexSchema, CommandSchema,
ParameterSchema, PrimaryKeySchema, TableKeySchema)</param>
/// <returns>An array of ExtendedProperties for a specific SchemaObjectBase.</returns>
public ExtendedProperty[] GetExtendedProperties(string connectionString, SchemaObjectBase
schemaObject)
{
    throw new NotImplementedException();
}

/// <summary>
/// Sets the extended properties.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="schemaObject">Any type that derives from SchemaObjectBase. (E.G. DatabaseSchema,
TableSchema, ColumnSchema, ViewSchema, ViewColumnSchema, IndexSchema, CommandSchema,
ParameterSchema, PrimaryKeySchema, TableKeySchema)</param>
public void SetExtendedProperties(string connectionString, SchemaObjectBase schemaObject)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets all of the tables available in the database.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="database">The database schema.</param>
/// <returns>An array of tables for a specific database.</returns>
public TableSchema[] GetTables(string connectionString, DatabaseSchema database)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets all columns for a given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>
/// <returns>An array of view columns for a specific table.</returns>
public ColumnSchema[] GetTableColumns(string connectionString, TableSchema table)

```

```

{
    throw new NotImplementedException();
}

/// <summary>
/// Gets all the views available for a given database.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="database">The database schema.</param>
/// <returns>An array of views for a specific database.</returns>
public ViewSchema[] GetViews(string connectionString, DatabaseSchema database)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the columns for a given view.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="view">The view schema.</param>
/// <returns>An array of view columns for a specific view.</returns>
public ViewColumnSchema[] GetViewColumns(string connectionString, ViewSchema view)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the definition for a given view.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="view">The view schema.</param>
/// <returns>The definition of a view.</returns>
public string GetViewText(string connectionString, ViewSchema view)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the primary key for a given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>
/// <returns>An the primary key for a specific table.</returns>
public PrimaryKeySchema GetTablePrimaryKey(string connectionString, TableSchema table)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets all of the table keys for a given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>
/// <returns>An array of keys for a specific table.</returns>
public TableKeySchema[] GetTableKeys(string connectionString, TableSchema table)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gats all of the indexes for a given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>

```

```

/// <returns>An array of indexes for a specific table.</returns>
public IndexSchema[] GetTableIndexes(string connectionString, TableSchema table)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the data from the given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>
/// <returns>A DataTable containing the data of the specific table.</returns>
public DataTable GetTableData(string connectionString, TableSchema table)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the data from a given view.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="view">The view schema.</param>
/// <returns>A DataTable containing the data of the specific view.</returns>
public DataTable GetViewData(string connectionString, ViewSchema view)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets all commands for the given database.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="database">The database schema.</param>
/// <returns>An array of commands.</returns>
public CommandSchema[] GetCommands(string connectionString, DatabaseSchema database)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the parameters for a given command.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="command">The command schema.</param>
/// <returns>An array of parameters.</returns>
public ParameterSchema[] GetCommandParameters(string connectionString, CommandSchema command)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets the definition for a given command.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="command">The command schema.</param>
/// <returns>The definition of a command.</returns>
public string GetCommandText(string connectionString, CommandSchema command)
{
    throw new NotImplementedException();
}

/// <summary>
/// Gets schema information about the results of a given command.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target

```

```

database.</param>
/// <param name="command">The command schema.</param>
/// <returns>An array of command results.</returns>
public CommandResultSchema[] GetCommandResultSchemas(string connectionString, CommandSchema
command)
{
    throw new NotImplementedException();
}

#endregion

```

```

#region Implementation of IDbConnectionStringEditor

public bool ShowEditor(string currentConnectionString)
{
    throw new NotImplementedException();
}

public string ConnectionString { get; private set; }
public bool EditorAvailable { get; private set; }

#endregion

```

### Implementing the SchemaExplorer Interfaces

It is highly recommended that you implement all of the methods and properties that were created. It is recommended to implement the ExtendedProperty methods but it isn't required. *In the future, we may provide a new abstract base class implementation which implements Extended Property support for you using a in memory database, but as of this time this is not on the official road map.*

The first section to implement would be the Name and Description properties which display the name and description of the Schema Provider in CodeSmith Generator.

```

/// <summary>
/// Gets the name of the schema provider (E.G. SqlSchemaProvider).
/// </summary>
public string Name { get { return "CustomSchemaProvider"; } }

/// <summary>
/// Gets the description for the schema provider (E.G. SQL Server Schema Provider)..
/// </summary>
public string Description { get { return "A Custom Schema Provider"; } }

```

Next, the GetDatabaseName method returns the name of a Database that is retrieved by the DataSource specified in the connection string (E.G., the Database name or a file name). From this point on, all of the methods are really important like the GetTables method

```

/// <summary>
/// Gets all of the tables available in the database.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="database">The database schema.</param>
/// <returns>An array of tables for a specific database.</returns>
public TableSchema[] GetTables(string connectionString, DatabaseSchema database)
{
    throw new NotImplementedException();
}

```

which returns all of the tables in this custom DataSource or the GetTableColumns which returns a list of columns for a specified table.

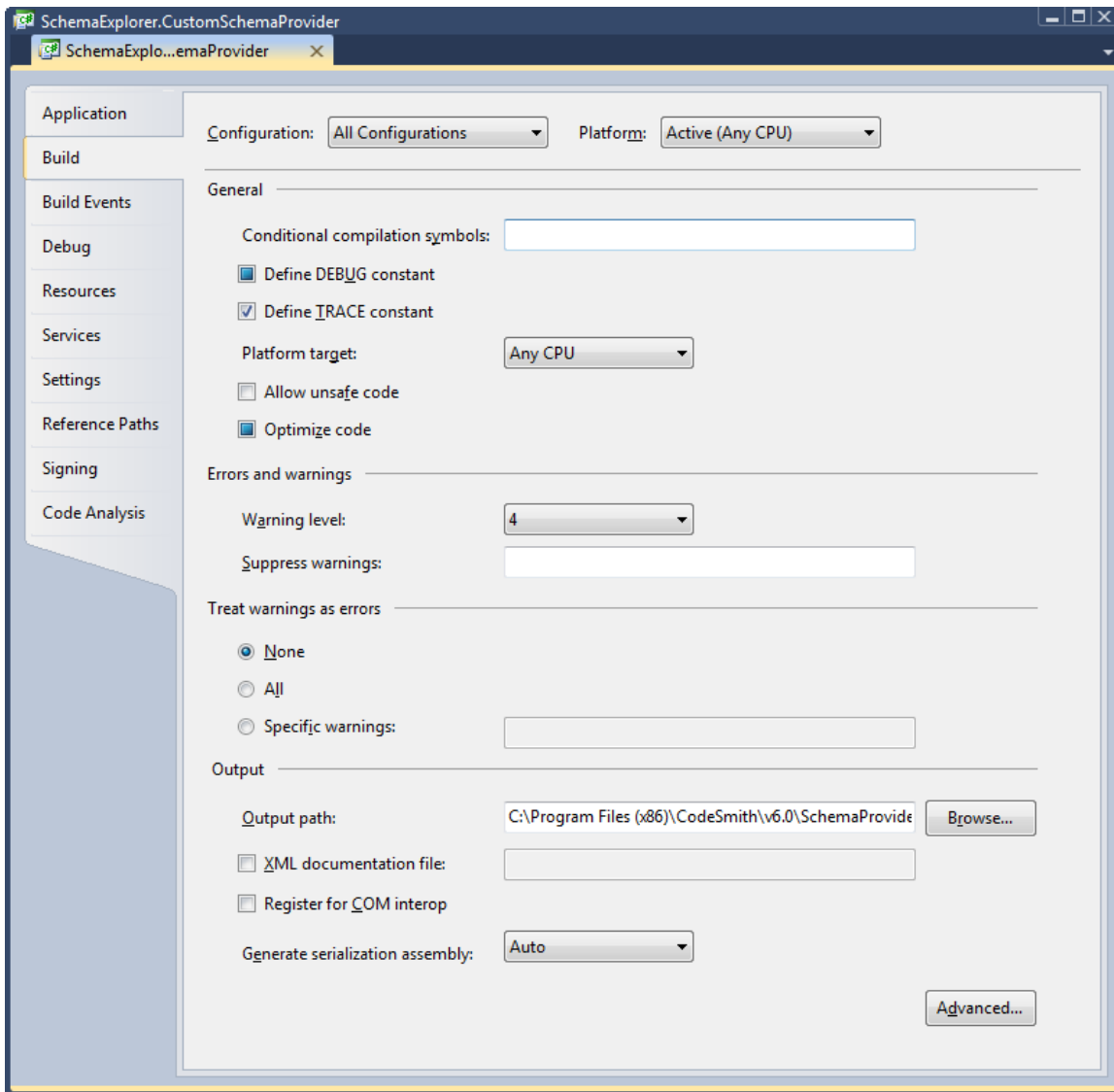
```
/// <summary>
/// Gets all columns for a given table.
/// </summary>
/// <param name="connectionString">The connection string used to connect to the target
database.</param>
/// <param name="table">The table schema.</param>
/// <returns>An array of view columns for a specific table.</returns>
public ColumnSchema[] GetTableColumns(string connectionString, TableSchema table)
{
    throw new NotImplementedException();
}
```

### **Download Custom Schema Provider Source**

Click [here](#) to download the source code above.

### **Building a Custom Schema Provider**

The first step is to set the output directory to the CodeSmith Generator Program Files Folder SchemaProviders Folder. To do this you will want to go into the Project's Properties page by right clicking the Project in Solution Explorer and selecting Properties. Next navigate to the Build tab and set the Configuration drop down list to **All Configurations**. *Please note that this may require that you run Visual Studio as an Administrator to build assemblies to a directory that requires elevated permissions via [User Account Control \(UAC\)](#). Also a build error may occur if CodeSmith Generator is open while building the Custom Schema Provider.*



You will want to set the Output path property value to the CodeSmith Generator Program Files Folder SchemaProviders Folder (C:\Program Files\CodeSmith\<Version>\SchemaProviders). You can do this by clicking on the browse button or manually typing the value in. It is recommended that you browse for this folder location. By setting this property it ensures that you will be building the Custom Schema Provider to the correct directory so CodeSmith Generator automatically picks up the latest changes when you restart CodeSmith Generator.

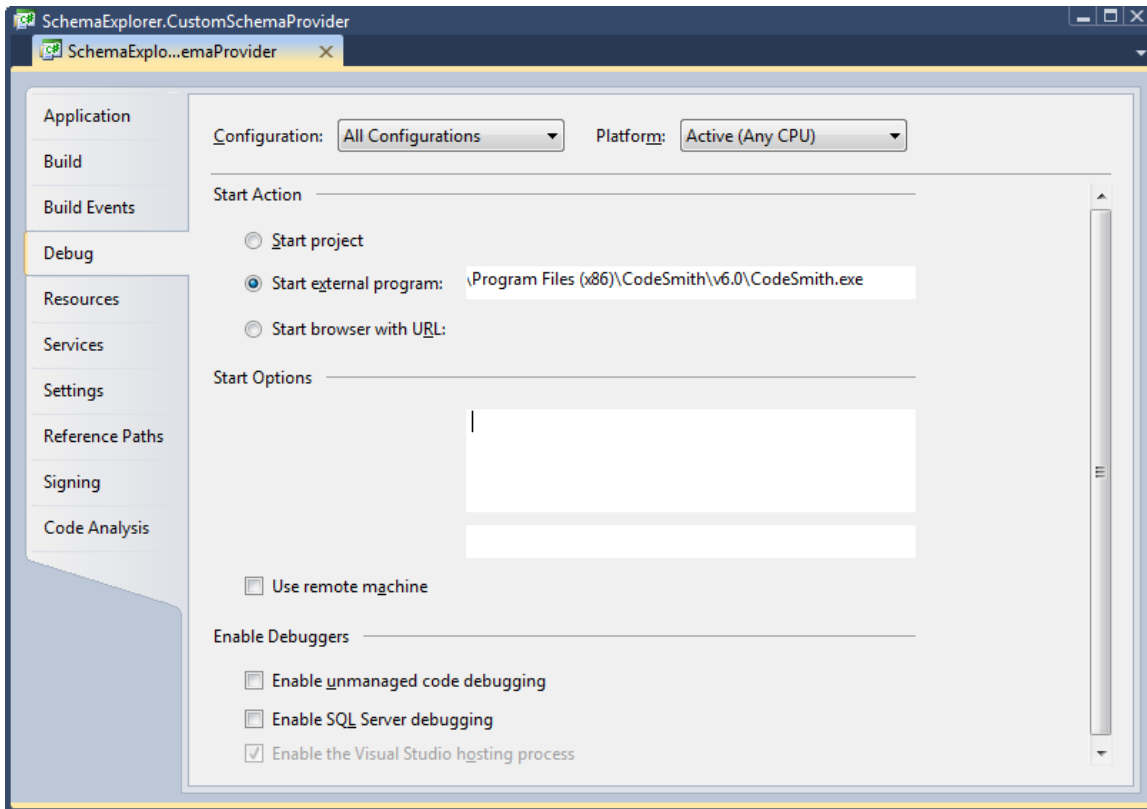
## Debugging a Custom Schema Provider

The first step to debugging a Custom Schema Provider is to make sure your project is set to compile in **Debug mode** as opposed to **Release mode**. The next step is to set a Start Up application so any of your breakpoints inside of your Custom Schema Provider get hit while running CodeSmith Generator. To do this you will want to go into the Project's Properties page by right clicking the Project in Solution Explorer and selecting Properties. Next navigate to the Debug tab and set the Configuration drop down list to **All Configurations**. Next under the Start Action Section choose the Start External Program. You will want to browse for the application you want to test the Custom Schema Provider with. In almost every case you will want to check out the Custom Schema Provider with CodeSmith Generator Explorer (C:\Program Files\CodeSmith\<Version>\CodeSmith.exe).

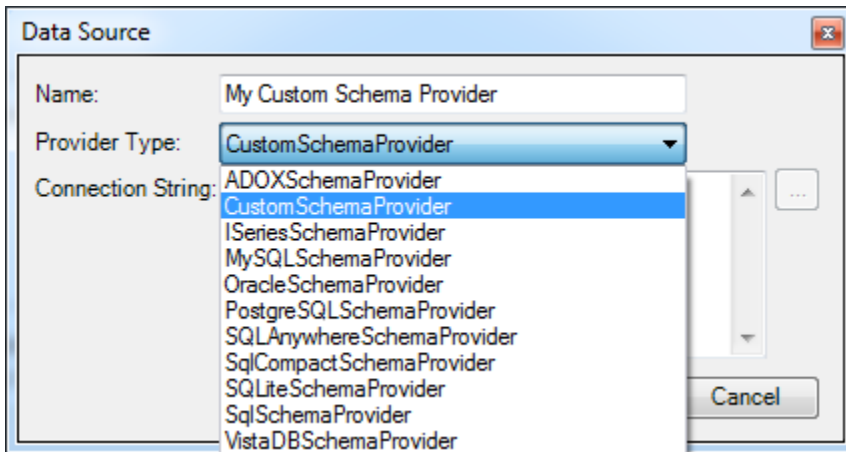


Please configure the Build Directory before debugging to ensure your breakpoints are hit and your latest changes take effect.





Next, set a breakpoint in your project and start debugging!



## Deploying a Custom Schema Provider

To deploy a custom Schema Provider you will need to:

- Make sure that the schema provider's assembly file name ends with "SchemaProvider.dll".
- Place the schema provider's compiled assembly (**bin\SchemaExplorer.CustomSchemaProvider.dll in our example**) in the CodeSmith Generator Program Files folder SchemaProviders Folder (ex. C:\Program Files\CodeSmith Generator\<Version>\SchemaProviders).
- In order to use the schema provider from Visual Studio, you will need to add the schema provider to the GAC.
- Restart CodeSmith Generator and/or Visual Studio.

## Upgrading a Custom Schema Provider

To upgrade an existing SchemaProvider to a newer version of CodeSmith Generator you will want to ensure that you remove all CodeSmith Generator project references and re-add them. Please note that you will want to ensure that they are pointing to the correct version of CodeSmith Generator by right clicking on the project reference and select properties. There will be Version information that is presented in the Visual Studio Properties Window. After this has been completed, just rebuild and redeploy to the CodeSmith Generator Program Files Folder SchemaProviders Folder.

## Using CodeSmith.CustomProperties

CodeSmith Generator ships with a sample project, CodeSmith.CustomProperties, that includes some useful property types for your templates:

- [FileNameEditor](#) can be used to let the user select a filename as the value for a property
- [StringCollection](#) can be used to let the user enter a collection of strings as the value for a property



You can find this project in your extracted samples folder (Documents\CodeSmith Generator\Samples<VERSION>\Projects\CSharp\CustomPropertiesSample).

### FileNameEditor

The `FileNameEditor` class lets you provide your users with a standard open file dialog box or save file dialog box from the CodeSmith Generator property grid. To use this class, you must include a reference to the `CodeSmith.CustomProperties` assembly in your template. You'll make the code simpler if you import the namespace as well:

```
<%@ Assembly Name="CodeSmith.CustomProperties" %>
<%@ Import Namespace="CodeSmith.CustomProperties" %>
```

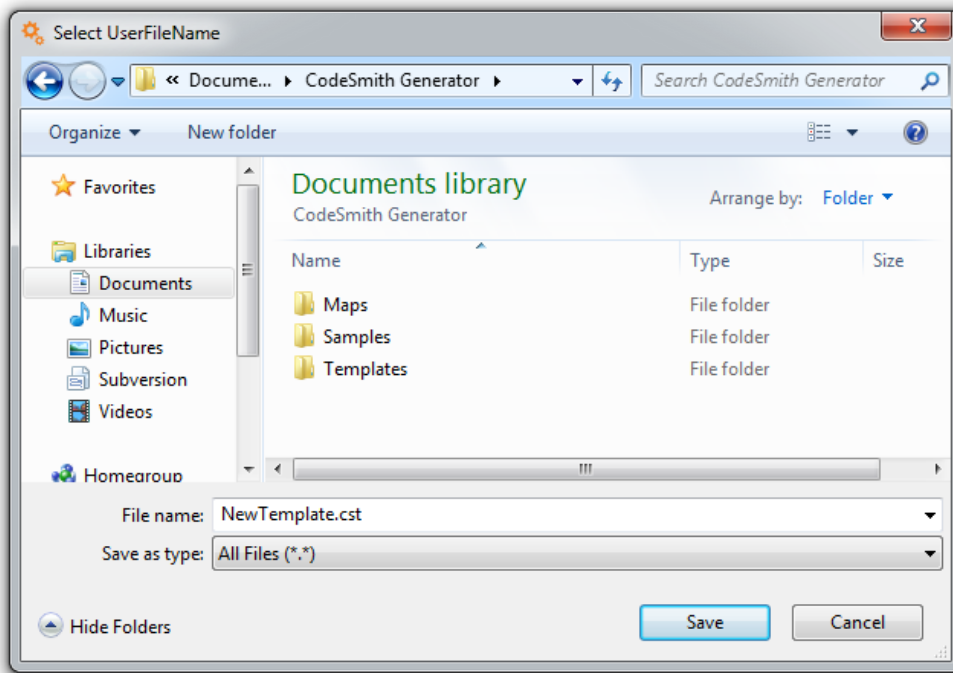
After you include the appropriate assembly reference, you can define a property in a script block that uses the `FileNameEditor`:

```
<script runat="template">
private string _userFileName = @"c:\temp\test.txt";
[Editor(typeof(FileNameEditor), typeof(System.Drawing.Design.UITypeEditor)),
Category("Custom"), Description("User selected file.")]
public string UserFileName
{
    get {return _userFileName;}
    set {_userFileName= value;}
}
</script>
```

When the user executes the template, the specified property will display a builder button on the property sheet (highlighted in green):



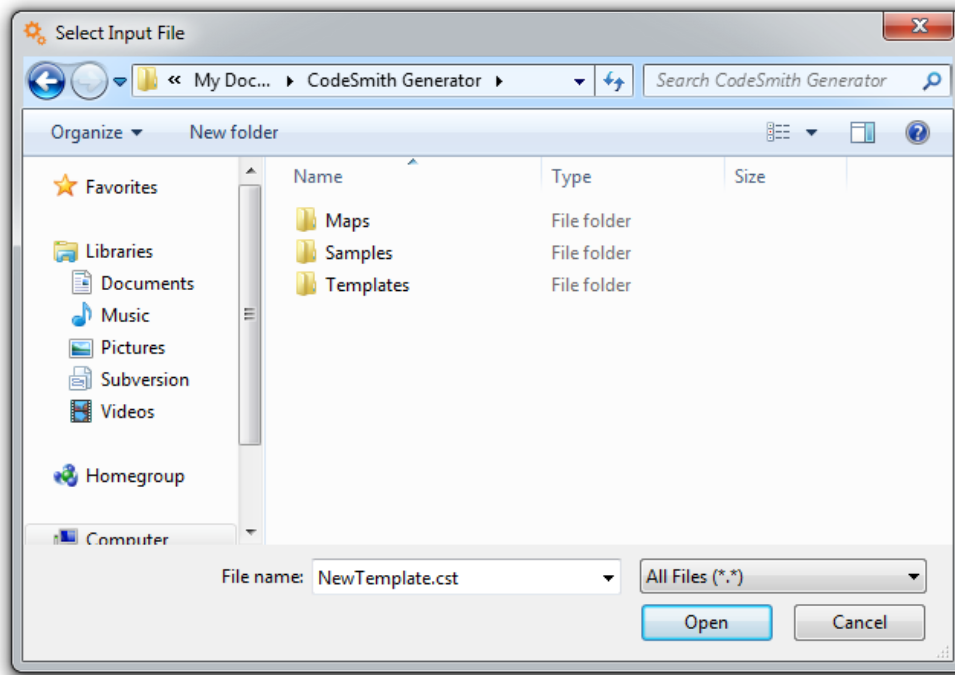
Clicking the builder button will open the specified file dialog box:



You can customize the appearance of the file dialog box by applying the `FileDialogAttribute` to the property. For example, consider this property definition:

```
private string _openFileName = @"c:\temp\test.txt";
[Editor(typeof(FileNameEditor), typeof(System.Drawing.Design.UITypeEditor)),
FileDialogAttribute(FileDialogType.Open, Title="Select Input File"),
Category("Custom"), Description("User selected file.")]
public string OpenFileName
{
    get {return _openFileName;}
    set {_openFileName= value;}
}
```

The File Dialog title has been updated in this example to "Select Input File" The resulting file dialog box looks like this:



You can specify these properties in the FileDialogAttribute:

| Property         | Meaning                                     | Default                    |
|------------------|---|----------------------------|
| FileDialogType   | Save or Open                                | FileDialogType.Save        |
| Filter           | Filter string for file extensions           | All Files (*.*)            |
| Title            | Dialog box title                            | Select <i>propertyname</i> |
| DefaultExtension | Default file extensions                     | None                       |
| CheckFileExists  | True to only allow selecting existing files | False                      |
| CheckPathExists  | True to only allow using existing paths     | False                      |

To select a folder name instead of a file name, use the FolderNameEditor class from the .NET Framework instead:

```

<%@ Assembly Name="System.Design" %>
<script runat="template">
private string _outputDirectory = @"c:\temp";
[Editor(typeof(System.Windows.Forms.Design.FolderNameEditor),
typeof(System.Drawing.Design.UITypeEditor)),
Category("Custom"), Description("Output directory.")]
public string OutputDirectory
{
    get {return _outputDirectory;}
    set {_outputDirectory= value;}
}
</script>

```

## StringCollection



Please take notice that this class has been marked as obsolete. Please use a generic collection instead.

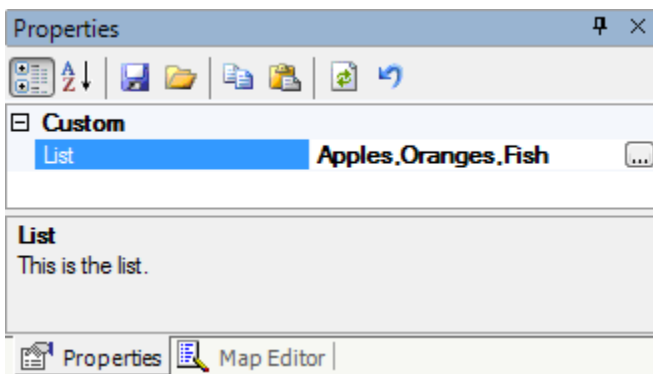
The StringCollection provides a way for users to enter a list of strings. In your code, you can refer to these strings as members of an array. To use this class, you must include a reference to the CodeSmith.CustomProperties assembly in your template:

```
<%@ Assembly Name="CodeSmith.CustomProperties" %>
```

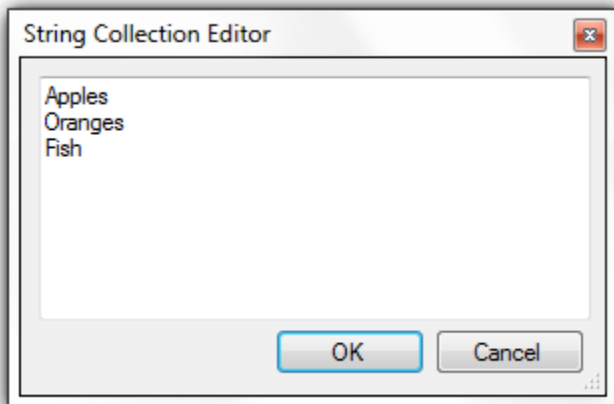
After you include the appropriate assembly reference, you can define a property in a script block that uses the StringCollection

```
<%@ Property Name="List" Type="CodeSmith.CustomProperties.StringCollection" Category="Custom"
Description="This is the list." %>
```

When the user executes the template, the specified property will display a builder button on the property sheet:



Clicking the builder button will open an editor that allows the user to type strings on separate lines:



You can also edit the members of the collection directly in the property grid as a comma-separated list.

In your code, you can iterate through the collection as an array:

The list is:

```
<% for (int i = 0; i < List.Count; i++)\{ %>
<%= List[i] %>
<% \} %>
```

## CodeSmith.BaseTemplates

The CodeSmith.BaseTemplates sample project contains two classes that inherit from CodeTemplate. These classes can be used to add functionality to your templates by referencing them in the Inherits attribute of the [CodeTemplate](#) directive:

- The [OutputFileCodeTemplate](#) class saves its output to a file
- The [SqlCodeTemplate](#) class includes utility functions for working with data stored in a SQL database

In addition, the project includes two utility classes that you can use via a reference to the CodeSmith.BaseTemplates assembly:

- The [StringUtil](#) class includes methods for working with strings
- The [ScriptUtility](#) class includes methods for working with SQL scripts



You can find this project in your extracted samples folder (Documents\CodeSmith Generator\Samples\<VERSION>\Projects\CSharp\BaseTemplates).

## OutputFileCodeTemplate

To base a template on the OutputFileCodeTemplate class, you inherit from this class in your template's [CodeTemplate](#) directive:

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Inherits="OutputFileCodeTemplate"
Description="Build custom access code." %>
<%@ Assembly Name="CodeSmith.BaseTemplates" %>
```

The OutputFileCodeTemplate class does two things. First, it adds a property named OutputFile to your template. This property requires you to select a filename. Second, the template overrides the [OnPostRender](#) method to write the output of your template to the specified file after CodeSmith Generator has finished generating code.



If you want to customize the Save File dialog box used by the OutputFile property, you can override the OutputFile property in your own template. For example, if you want to force the user to select a .cs file for output, you'd include this code in your template:

```
<script runat="template">
// Override the OutputFile property and assign our specific settings to it.
[FileDialog(FileDialogType.Save, Title="Select Output File", Filter="C# Files (.cs)|.cs",
DefaultExtension=".cs")]
public override string OutputFile
{
    get {return base.OutputFile;}

    set {base.OutputFile = value;}
}
</script>
```

## SqlCodeTemplate

To base a template on the SqlCodeTemplate class, you inherit from this class in your template's [CodeTemplate](#) directive:

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" Inherits="SqlCodeTemplate" Description="Build data access layer." %>
<%@ Assembly Name="CodeSmith.BaseTemplates" %>
```

The `SqlCodeTemplate` class contains numerous utility methods designed to make it easier to work with SQL databases. These include:

- **GetCSharpVariableType** - Returns the equivalent C# variable type for a database column.
- **GetMemberVariableDeclarationStatement** - Returns a C# member variable declaration statement.
- **GetMemberVariableDefaultValue** - Returns a default value based on a column's data type.
- **GetMemberVariableName** - Returns the C# member variable name for a given identifier.
- **GetPropertyNames** - Returns the name of the public property for a given column.
- **GetReaderMethod** - Returns the name of the typed reader method for a given column.
- **GetSqlDbType** - Returns the `SqlDbType` based on a given column.
- **GetSqlParameterExtraParams** - Generates any extra parameters that are needed for the ADO parameter statement.
- **GetSqlParameterStatement** - Returns a T-SQL parameter statement based on the given column.
- **GetSqlParameterStatements** - Generates an assignment statement that adds a parameter to a ADO object for the given column.
- **GetSqlReaderAssignmentStatement** - Returns a typed C# reader.`ReadXXX()` statement.
- **GetValidateStatements** - Generates a batch of C# validation statements based on the column.
- **IncludeEmptyCheck** - Determines if a given column should use a check for an Empty value.
- **IncludeMaxLengthCheck** - Determines if the given column's data type requires a maximum length to be defined.
- **IsUserDefinedType** - Determine if the given column is using a UDT.



For a complete listing of all `SqlCodeTemplate` methods please refer to [the API documentation](#).

## StringUtil

To use the functions in the `StringUtil` class, you should set a reference to the `CodeSmith.Engine` assembly and import its namespace:

```
<%@ Assembly Name="CodeSmith.Engine" %>
<%@ Import Namespace="CodeSmith.Engine" %>
```

The `StringUtil` class includes these static methods:

- **IsPlural** - returns True if a string is plural.
- **IsSingular** - returns True if a string is singular
- **ToCamelCase** - converts a set of words to a single camel case identifier
- **ToPlural** - converts a word to its plural form
- **ToSingular** - converts a word to its singular form
- **ToSpacedWords** - converts a camel case identifier to separate words



For a complete listing of all `StringUtil` methods please refer to [the API documentation](#).

## Example

The following example will show how you can mapping overrides and `StringUtil`. `StringUtil` `ToPlural` and `ToSingular` supports overriding the converted word with a special `csmmap` file. The default override file is called `Plural-Overrides.csmmap`. It will be used by default. However, you can use a different `Map` file if needed.

```
<%@ CodeTemplate Language="C#" TargetLanguage="Text"
    Debug="False" Description="Plural Overrides." %>

<%@ Map Name="PluralOverrides"
    Src="Plural.csmmap" Reverse="False"
    Description="Convert system data types to c# alias" %>
<%@ Map Name="SingleOverrides"
    Src="Plural.csmmap" Reverse="True"
    Description="Convert system data types to c# alias" %>
```

### Using the built in csmap

```
Child: <%= StringUtil.ToPlural("Child") %>  
Children: <%= StringUtil.ToSingular("Children") %>
```

### Using the MapCollection

```
Knife: <%= StringUtil.ToPlural("nife", PluralOverrides) %>  
Knives: <%= StringUtil.ToSingular("nives", SingleOverrides) %>
```

### Using the csmap name

```
Knife: <%= StringUtil.ToPlural("nife", "Plural.csmap") %>  
Knives: <%= StringUtil.ToSingular("nives", "Plural.csmap") %>
```

## ScriptUtility

To use the functions in the ScriptUtil class, you should set a reference to the CodeSmith.Engine assembly and import its namespace:

```
<%@ Assembly Name="CodeSmith.Engine" %>  
<%@ Import Namespace="CodeSmith.Engine" %>
```

The ScriptUtil class includes a single static method to execute a SQL script against a supplied connection. You'll find this method useful when you want to [execute generated SQL scripts](#).

## Building a custom UITypeEditor

This document will walk you through the process of building a custom UITypeEditor as shown [here](#). This UITypeEditor will populate a DropDownList with table names from a chosen database.

### Building the DropDownListProperty

First you need to create a public class that will hold the data of the drop down list. In this example I named my class DropDownListProperty.



```
public class DropDownListProperty
{
}
```

Next we will need to add the properties and the constructors.

```
public class DropDownListProperty
{
    private List<string> _values = new List<string>();

    public DropDownListProperty()
    {
        SelectedItem = "None";
    }

    public DropDownListProperty(List<String> values)
    {
        if(values.Count > 0)
            SelectedItem = values[0];
        else
            SelectedItem = "None";

        Values = values;
    }

    public List<string> Values
    {
        get
        {
            if (_values == null)
                _values = new List<String>();

            return _values;
        }
        set
        {
            if(value != null)
                _values = value;
        }
    }

    [Browsable(false)]
    public string SelectedItem { get; set; }
}
```

You'll notice that we have a public property called SelectedItem. This property will hold the initial value which will be the selected value when a user selects a choice. By default we set this to "None" in the constructor. We also set an attribute on the property Browsable(false). This tells the PropertyGrid not to display this property.

### ***Updating the content that is displayed in the Property Sheet.***

We now want to override the ToString() method to the DropDownListProperty so the dropdown displays the current selected value.

```
/// <summary>
/// The value that we return here will be shown in the property grid.
/// </summary>
/// <returns></returns>
public override string ToString()
{
    return SelectedItem;
}
```

### ***Implementing the custom UITypeEditor***

Now it is time to implement the class that controls how my class is displayed in the property grid. We will want to create a class that inherits from `UITypeEditor`.

```
/// <summary>
/// Provides a user interface for selecting a state property.
/// </summary>
public class DropDownListPropertyEditor : UITypeEditor
{
}
```

Next we will add a private member variable named `_service`. We will need to declare this member variable because we will want to tie into an event in a little bit. Now it is time to override the `EditValue` method.

```
/// <summary>
/// Displays a list of available values for the specified component than sets the value.
/// </summary>
/// <param name="context">An ITypeDescriptorContext that can be used to gain additional context
information.</param>
/// <param name="provider">A service provider object through which editing services may be
obtained.</param>
/// <param name="value">An instance of the value being edited.</param>
/// <returns>The new value of the object. If the value of the object hasn't changed, this method
should return the same object it was passed.</returns>
public override object EditValue(ITypeDescriptorContext context, IServiceProvider provider, object
value)
{
    if (provider != null)
    {
        // This service is in charge of popping our ListBox.
        _service =
            ((IWindowsFormsEditorService)provider.GetService(typeof(IWindowsFormsEditorService)));

        if (_service != null && value is DropDownListProperty)
        {
            var property = (DropDownListProperty) value;

            var list = new ListBox();
            list.Click += ListBox_Click;

            foreach (string item in property.Values)
            {
                list.Items.Add(item);
            }

            // Drop the list control.
            _service.DropDownControl(list);

            if (list.SelectedItem != null && list.SelectedIndex.Count == 1)
            {
                property.SelectedItem = list.SelectedItem.ToString();
                value = property;
            }
        }
    }

    return value;
}
```

It is important not to be overwhelmed by the code above. The object value that is passed in is the `DropDownListProperty` class that holds our data. All we need to do is some safe type checking (`value is DropDownListProperty`) and then cast the value. The `_service` variable holds the property grid control that we are interacting with.

We create a `ListBox` object as that will hold our list of data (`Values` property from the `DropDownListProperty` class). It also exposes a `Click` event that will allow us to know when someone clicks on the drop down list. We will add an event handler `ListBox_Click` to the `Click` event so we can close the drop down list. If we skipped this step then the list would always be shown.

The next few lines just adds all our data into the Listox and calls DropDownControl(Control). This shows the populated ListBox control.

Finally we will set the SelectedItem to the Item that the user selected.

It is time to add the method that we wired up to the Click event.


```
private void ListBox_Click(object sender, EventArgs e)
{
    if(_service != null)
        _service.CloseDropDown();
}
```


The last piece to this puzzle is to override the GetEditStyle method and return that we want to display a DropDown UITypeEditorEditStyle

```
/// <summary>
/// Gets the editing style of the <see cref="EditValue"/> method.
/// </summary>
/// <param name="context">An ITypeDescriptorContext that can be used to gain additional context
information.</param>
/// <returns>Returns the DropDown style, since this editor uses a drop down list.</returns>
public override UITypeEditorEditStyle GetEditStyle(ITypeDescriptorContext context)
{
    // We're using a drop down style UITypeEditor.
    return UITypeEditorEditStyle.DropDown;
}
```

Finally we will go back and add a Editor attribute to the DropDownListProperty class. This will tell the PropertyGrid that when this property type is loaded to use the new UITypeEditor class we created.

```
[Editor(typeof(DropDownListPropertyEditor), typeof(System.Drawing.Design.UITypeEditor))]
```

 Please click [here](#) for the complete source code for these two classes.

 For more information on building custom UITypeEditors, please refer to the fol Michael Weinhardt and Chris Sells' article "Building Windows Forms Controls and Components with Rich Design-Time Features, Part 2" on the MSDN Web site.

## Setting up a DataDirectory for Generator Connection Strings

### Using a DataDirectory for Generator

You can specify a DataDirectory for CodeSmith Generator to easily share and discover MS SQL Express databases. CodeSmith Generator ships with a version of the Petshop database whose Datasource uses a DataDirectory.

#### **The DataDirectory Path**

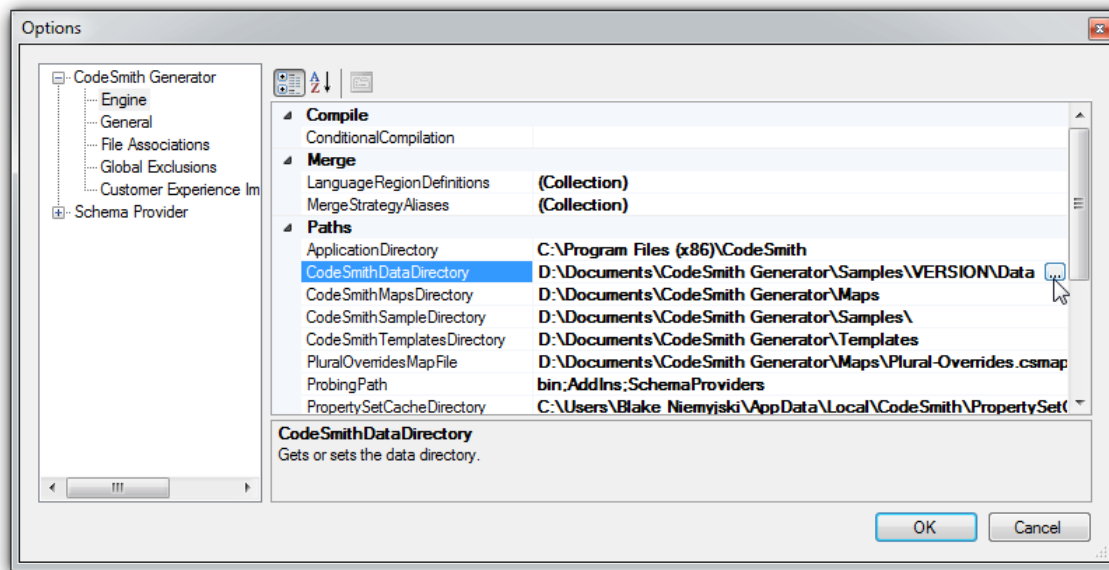
By default, the DataDirectory folder for CodeSmith Generator is:

**Windows 2000/XP:** C:\Documents and Settings\<USER NAME>\My Documents\CodeSmith Generator\Samples\<VERSION>\Data

**Windows Vista/Windows 7:** C:\Users\<USER NAME>\Documents\CodeSmith Generator\Samples\<VERSION>\Data

#### **Customizing the DataDirectory Path**

You can customize the path that Generator uses to set up the DataDirectory in the AppDomain. The first step is to open the Generator Options dialog. Once this dialog is open, select the Engine node on the left hand side of the options.



Finally, look for the CodeSmithDataDirectory Property under the Paths category and select the folder picker. Changing this path to another directory will cause Generator to use the new path the next time Generator starts up.

## Frequently Asked Questions

### How can I add comments to my templates?

Template comments can be added using the `<%- -%>` tokens.

### How can I create a property that has a drop-down of values to choose from?

Use a script block to define an [enumerated property](#).

### How can I prevent ASP.NET tags from confusing my templates?

Properly escape them using ASP.NET tags.

### How can I declare a constant in my template?

Constants must be declared inside `<script runat="template">` tags.

```
<script runat="template">
private const string MY_CONST = "example";
</script>
```

### How can I debug my templates?

You can compile your templates in [debug mode](#) and set breakpoints in them.

### How can I add a property that lets me select a folder?

Decorate the property with the [FolderNameEditor](#) attribute.

### How can I use sub-templates?

CodeSmith Generator includes a full API to let you make use of [sub-templates](#).

### What assemblies and namespaces are loaded by default into a template?

### Assemblies:

- System
- System.Core
- System.Xml
- System.Data
- System.Drawing
- System.Design
- Microsoft.VisualBasic
- System.Windows.Forms
- CodeSmith.Engine
- CodeSmith.Core

### Namespaces:

- System
- System.Data
- System.Diagnostics
- System.ComponentModel
- Microsoft.VisualBasic
- CodeSmith.Engine

### Is it possible to determine whether a column is an identity column using SchemaExplorer?

Yes, use the [CS\\_IsIdentity](#) property.

### Is it possible to determine a column's default value using SchemaExplorer?

Yes, use the [CS\\_Default](#) property.

```
<%
    foreach(ColumnSchema cs in SourceTable.Columns)
    {
        if (cs.ExtendedProperties["CS_Default"] != null)
        {
            Response.WriteLine(cs.ExtendedProperties["CS_Default"].Value);
        }
    }
%>
```

### How can I enumerate the input and output parameters of a stored procedure using SchemaExplorer?

The [CommandSchema](#) object contains both input and output parameter collections which can be used to read these parameters.

### What if my template contains non-ASCII characters?

You can use the [ResponseEncoding](#) attribute of the [CodeTemplate](#) directive to set the encoding for the template.

## Tips and Tricks

- When you need to generate multiple files as part of a single code-generation process, consider using one sub-template for each file. Call the sub-templates from a master template and use the [RenderToFile](#) method to output each sub-template.
- To generate multiple output files as part of an automated process, you can use the [CodeSmith Generator Project](#) support from the console or MSBuild.
- You can use [Template Explorer](#) to create property set XML files for use by the [CodeSmith Generator Console Application](#).
- You can drag and drop a template from [Template Explorer](#) window to any application that supports dropping text. When you drop the template, CodeSmith Generator will display the template's property sheet. Fill in the metadata for the template, click the Generate button, and the template's output will be rendered directly to the application where you dropped the template.
- The new [XmlPropertydirective](#) gives you a strongly-typed object model and statement completion for metadata stored in XML files.
- You can mix your own custom code with generated code by employing one of several different [active generation strategies](#).
- If you're generating SQL Scripts, CodeSmith Generator can [execute the scripts](#) for you immediately after generating them.
- Use [merge strategies](#) to preserve custom code while regenerating from templates as part of an automated build process.
- Use [Ctrl+Shift+M](#) in the Template Editor to collapse all template code blocks. This is useful to quickly see all static template content.
- Use the [StoredProcedures.cst](#) template to generate standard create, retrieve, update, and delete stored procedures for a given table instantly.
- You can use [Trace.WriteLine](#) to output messages to the [Debug](#) output window.

## Internet Links

- [Home Page](#)
- [Sales](#)
- [Support](#)
- [CodeSmith Community](#)
- [Template Gallery](#)

## Reference

For references please see:

- [System Requirements](#)
- [CodeSmith Generator Samples](#)

## System Requirements

CodeSmith Generator requires the .NET Framework (version 4.0), but it has no other hardware or software prerequisites. It should run fine on any system that meets Microsoft's [minimum requirements](#) for installing the .NET Framework.

## CodeSmith Generator Samples

When you install CodeSmith Generator, you also get a wide variety of useful samples. These samples are of two types. Sample templates, located under the `Documents\CodeSmith Generator\Samples\<VERSION>\Projects\Templates` folder, can be loaded into CodeSmith Generator and used to generate code. Sample projects, located under the `Documents\CodeSmith Generator\Samples\<VERSION>\Projects\Samples` folder, show you how you can extend and customize CodeSmith Generator.

### Generator Templates

CodeSmith Generator ships with a complete template set that helps you get up and running in no time flat. These template sets include ActiveSnippet Templates, Database Templates, Example Templates, [Framework Templates](#) and various other templates. Why waste time with repetitive tasks? Use ActiveSnippets and focus on other parts of your application.

### *ActiveSnippet Templates*

[ActiveSnippet's](#) allow you to quickly reduce the amount of time it takes to get your job done. Any template can be used as an ActiveSnippet. CodeSmith Generator ships with active snippets that will speed up the process of creating custom events and exceptions. Also you can quickly generate an enumeration or properties from database meta data.

### *Database Templates*

Whether you are looking for an easier ways to create a business object, document your database, script your table data, execute or create stored procedures, generator is here to save you time and headaches.

The **Business Object template** is a template that is a great template to use if your looking to quickly create an entity for your project. Many developers also use this template as a base template when they need to create a new POCO (Plain Old CLR Object), Data Transfer Object or Domain Class template.

The **Script Table Data and Extended Properties template** allows you to generate a SQL script that can be used to migrate your data between database servers or import existing data into newly created databases.

The **DbDocumenter template** will create a nicely formatted html document of your entire database.

The **Stored Procedure template** will create (Insert, Update, Delete, Select) Stored Procedures based on a Database Table. Everything is configurable to fit your unique requirements.

The **Command Wrapper template** will create a wrapper around any SQL Stored Procedure or SQL Function. An easy to use API allows you to execute a stored procedure or function by calling `Execute()` which will return get back a strongly typed object or result. When used in conjunction with the Stored Procedure templates, you can quickly get data from your database.

The **Typed DataSet template** creates typed DataSet and DataAdapter classes based on a database table of your choosing.

### *Example Templates*

We provide a set of example templates that demonstrate how to use various [Generator features](#) in your own custom template.

The **ASP.NET folder** contains a template that will show you how to use Master Templates that will generate an ASP.NET default.aspx page. (Watch the [Master Templates Video](#))

The **Basic Samples folder** contains templates that display the use of Master Templates, Partial templates and LinQ To Objects in your templates.

The **Maps folder** is a great example on how and when to use [CodeSmith Generator Maps](#).

The **Merge folder** contains templates that will show you how to use Preserve Region and Insert Region Merge Strategies across different file types and languages. (Watch the [Merge Strategies Video](#))

The **Photo Gallery folder** contains templates combine using a Master Templates and code behind to create a generic photo gallery from a directory of image files.

The **Xml folder** contains templates that show you how to generate from an Xml data source using the XmlProperty. (Watch the [XmlProperty Video](#))

### **Other Templates**

The Other templates folder contains various templates that ease the use working with AJAX, Custom Collections (ArrayList, HashTable, Queues, SortedList and more) and [WIX \(Windows Installer Xml toolset\)](#).

### **Generator Sample Projects**

**SchemaProvider source code** - All [schema provider](#) source code can be found under the CSharp folder.

**BaseTemplates** - This project includes the source code for the [CodeSmith.BaseTemplates](#) classes.

**ConsoleSamples** - This folder contains samples for use with the [CodeSmith Console Application](#).

**VSIntegrationSample** - This project contains a sample of using the VS.NET integration to simulate generics with the [CodeSmith Generator Project](#) integration.

**CustomPropertiesSample** - This project includes the source code for the [CodeSmith.CustomProperties](#) classes.

**APISample** - This project demonstrates the use of the [CodeSmith API](#).

**TypedDataSetSample** - This project contains a test application for the typed DataSet templates.

## **Licensing and Distribution**

For Licensing and Distribution information please see:

- [Copyrights and Trademarks](#)
- [Software Licenses](#)
- [CodeSmith Generator Editions](#)
- [Product Activation / Deactivation](#)

## **Copyrights and Trademarks**

CodeSmith Generator is a trademark of CodeSmith Tools, LLC.

.NET, Visual Basic, and Visual C# are either registered trademarks or trademarks of [Microsoft Corporation](#) in the United States and/or other countries.

## **Software Licenses**

### **CODESMITH GENERATOR LICENSE AGREEMENT**

**IMPORTANT--READ CAREFULLY:** This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or an entity) and CodeSmith Tools, LLC ("CODESMITH") for the CodeSmith Generator template based code generator software product and any included components or materials ("SOFTWARE PRODUCT"). BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE PRODUCT, YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA. IF YOU DO NOT AGREE TO THE TERMS OF THIS EULA, YOU ARE NOT AUTHORIZED TO INSTALL, COPY, OR OTHERWISE USE THE SOFTWARE PRODUCT.

#### **SOFTWARE PRODUCT LICENSE**

The SOFTWARE PRODUCT is protected by United States copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. GRANT OF LICENSE. This EULA grants you the following rights:

a. Per User License. The SOFTWARE PRODUCT permits a single user to use the SOFTWARE PRODUCT under the terms of the license. If the SOFTWARE PRODUCT is installed on a single computer used by multiple users, a customer must purchase additional licenses for each user that accesses the SOFTWARE PRODUCT. Further, if the SOFTWARE PRODUCT is installed or accessed through a network, the customer must

purchase additional licenses for each user that accesses the SOFTWARE PRODUCT through the network.

b. Use of Generated Output. You may distribute the output of your custom templates or the included templates in any way.

2. COPIES. You may make copies of the SOFTWARE PRODUCT provided that any such copy: (i) is created as an essential step in the utilization of the SOFTWARE PRODUCT as licensed under this EULA, or (ii) is only used for archival purposes to back-up the Software. All trademark, copyright and proprietary rights notices must be faithfully reproduced and included by you on such copies. You may not make any other copies of the Software.

### 3. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

a. Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

b. Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

c. Redistribution. The SOFTWARE PRODUCT may not be redistributed in any way.

d. Custom Template Distribution. You may distribute your custom templates for the SOFTWARE PRODUCT in any way that you like. You may also charge money for your custom templates.

e. No Rental. You may not rent, lease, lend or provide commercial hosting services to third parties with the SOFTWARE PRODUCT.

f. Termination. Without prejudice to any other rights, CODESMITH may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

### 4. ADDITIONAL SOFTWARE/SERVICES.

a. Supplements. This EULA applies to additional software and updates of the SOFTWARE PRODUCT, including without limitation supplements, service packages, hot fixes, or add-on components (collectively "Supplements") that CODESMITH may provide to you or make available to you after the date you obtain your initial copy of the SOFTWARE PRODUCT, unless other terms are provided along with such Supplements.

5. COPYRIGHT. All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, SAMPLE CODE, and "applets" incorporated into the SOFTWARE PRODUCT) and any copies of the SOFTWARE PRODUCT are owned by CODESMITH. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material except that you may install the SOFTWARE PRODUCT.

6. U.S. GOVERNMENT RESTRICTED RIGHTS. All SOFTWARE PRODUCT provided to the U.S. Government pursuant to solicitations issued on or after December 1, 1995 is provided with the commercial license rights and restrictions described elsewhere herein. All SOFTWARE PRODUCT provided to the U.S. Government pursuant to solicitations issued prior to December 1, 1995 is provided with "Restricted Rights" as provided for in FAR, 48 CFR 52.227-14 (JUNE 1987) or DFAR, 48 CFR 252.227-7013 (OCT 1988), as applicable.

7. EXPORT RESTRICTIONS. You acknowledge that the SOFTWARE PRODUCT is subject to U.S. export jurisdiction. You agree to comply with all applicable international and national laws that apply to these products, including the U.S. Export Administration Regulations, as well as end-user, end-use and destination restrictions issued by U.S. and other governments.

### 8. DISCLAIMER OF WARRANTY

To the maximum extent permitted by applicable law, CODESMITH provides the SOFTWARE PRODUCT and support services (if any) AS IS AND WITH ALL FAULTS, and hereby disclaim all other warranties and conditions, whether express, implied, or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of reliability or availability, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence, all with regard to the SOFTWARE PRODUCT, and the provision of or failure to provide support or other services, information, software, and related content through the SOFTWARE PRODUCT or otherwise arising out of the use of the SOFTWARE PRODUCT. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION, OR NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE PRODUCT.

### 9. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL, AND CERTAIN OTHER DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL CODESMITH BE LIABLE FOR ANY SPECIAL, INCIDENTAL, PUNITIVE, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE PRODUCT OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE PRODUCT, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, BREACH OF CONTRACT, OR BREACH OF WARRANTY OF CODESMITH, AND EVEN IF CODESMITH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

10. LIMITATION OF LIABILITY AND REMEDIES. NOTWITHSTANDING ANY DAMAGES THAT YOU MIGHT INCUR FOR ANY REASON WHATSOEVER (INCLUDING, WITHOUT LIMITATION, ALL DAMAGES REFERENCED ABOVE AND ALL DIRECT OR GENERAL DAMAGES), THE ENTIRE LIABILITY OF CODESMITH UNDER ANY PROVISION OF THIS EULA AND YOUR EXCLUSIVE REMEDY FOR ALL OF THE FOREGOING SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S.\$5.00. THE FOREGOING LIMITATIONS, EXCLUSIONS AND DISCLAIMERS SHALL APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, EVEN IF ANY REMEDY FAILS ITS ESSENTIAL PURPOSE.

11. APPLICABLE LAW. This EULA is governed by the laws of the State of Texas.

12. ENTIRE AGREEMENT. This EULA (including any addendum or amendment to this EULA which is included with the SOFTWARE PRODUCT) is the entire agreement between you and CODESMITH relating to the SOFTWARE PRODUCT and support services (if any), and it supersedes all



prior or contemporaneous oral or written communications, proposals, and representations with respect to the SOFTWARE PRODUCT or any other subject matter covered by this EULA. To the extent the terms of any CODESMITH policies or programs for support services conflict with the terms of this EULA, the terms of this EULA shall control.

## Premier Support

### Why should I purchase Premier Support?

Premier Support is a support option that can be added to any of our CodeSmith Generator licenses. With it you get the following:

- FREE upgrades to new major and minor releases of CodeSmith Generator.
- Priority access to the CodeSmith development team for your support issues.
- 1 Business day email response time with priority over standard users.
- Access to bug fixes and nightly builds without waiting for official releases.

Those without current Premier Support will only have access to the version they received at the time of purchase and will have to pay for any major upgrades.

### What is the cost of Premier Support?

The cost of Premier Support is \$99 for one full year and at the end of the year we send out a renewal notice to remind you of when your support expires so you can renew it before the end of the expiration date. Upon renewing the license, we add the year you paid for to the end of the expiration date.

### When can I purchase Premier Support?

Users have the following choices:

- Purchase 1 year of Premier Support at the time of purchasing or upgrading a license.
- Purchase 1 year of Premier Support at a later date. This will include an additional fee on top of the Premier Support cost.
- Pay for support on a per incident basis.

You can purchase Premier Support by following the steps below.

### Where can I purchase Premier Support?

#### New Orders

You can add Premier Support to your cart when adding CodeSmith Generator to your [cart](#).


#### Existing Orders

You can purchase Premier Support by visiting the [Premier Support Renewal Form](#) and entering your CodeSmith Generator license key.

## CodeSmith Generator Editions


### CodeSmith Generator

CodeSmith Generator includes batch code generation, template caching, Visual Studio integration, the ability to use the CodeSmith API in custom internal applications, merging support and much more. From small to large complex code generation scenarios CodeSmith Generator is the perfect tool.

 CodeSmith Generator is licensed per-user.

### CodeSmith Generator SDK

The SDK Edition enables distribution of custom applications that use the Generator API (including SchemaExplorer). These applications can make use of the full power of the Generator Engine in a programmatic fashion. You will be able to distribute the Generator assemblies with your application and include a runtime SDK license either in your application folder or shared license folder.

 The Generator SDK only allows for the distribution of Generator Assemblies internally. To use the Generator SDK outside of your organization one must purchase a Commercial Generator SDK license.

### CodeSmith Generator Server

The Server Edition allows you to install CodeSmith Generator on a Build Server or Server. These applications can make use of the full power of the Generator Engine in a programmatic fashion. The Server Edition does not allow any access to CodeSmith Generator's User Interfaces like Template Explorer.

**CodeSmith Generator Features:**

| Feature                             | Generator | Generator SDK | Generator Server |
|-------------------------------------|-----------|---------------|------------------|
| Simple Template Syntax              |           |               |                  |
| Execute Custom Templates            |           |               |                  |
| Auto SQL Script Execution           |           |               |                  |
| Extensible Metadata                 |           |               |                  |
| SchemaExplorer Schema Discovery API |           |               |                  |
| Rich XML Support                    |           |               |                  |
| Sub Template Support                |           |               |                  |
| Useful Sample Templates             |           |               |                  |
| Console Client                      |           |               |                  |
| Template Explorer Client            |           |               |                  |
| Generator Template Editor           |           |               |                  |
| Generator Map Support               |           |               |                  |
| Visual Studio Integration           |           |               |                  |
| Generator API                       |           |               |                  |
| Statement Completion                |           |               |                  |
| Template Caching                    |           |               |                  |
| Template Debugging                  |           |               |                  |
| Merge Capabilities                  |           |               |                  |
| Generator Project Support           |           |               |                  |
| MSBuild Support                     |           |               |                  |
| ActiveSnippet Support               |           |               |                  |

**Product Activation and Deactivation**

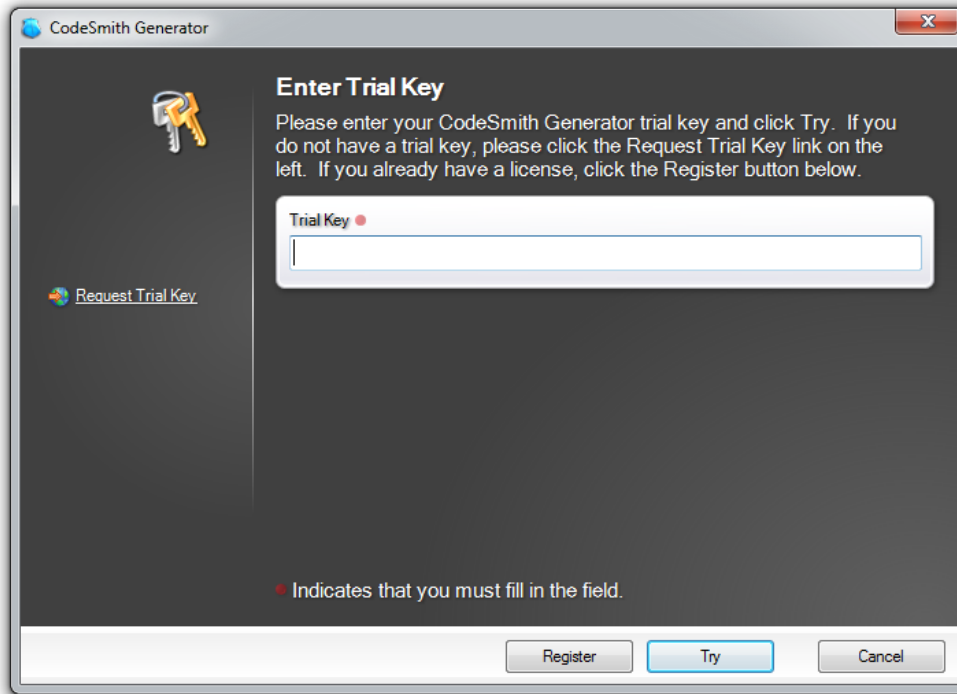
**Activation**


CodeSmith Generator must be activated on each computer where you will use it. Activation issues a unique license code to the computer to permanently unlock CodeSmith Generator's functionality. The activation technology that CodeSmith Generator uses is designed to be as painless as possible, and to never lock you out from your legitimate use of the product. By default, you can activate CodeSmith Generator on three different computers, but you can obtain additional activations simply by sending an e-mail to [support@codesmithtools.com](mailto:support@codesmithtools.com). When you install a new copy of CodeSmith Generator, you have a 30-day trial period to register the product, and then another 30-day grace period to complete the product activation process.


When you first launch [Template Explorer](#) or the [Template Editor](#), CodeSmith Generator will display the Enter Registration Information dialog box:

## Trial Activation

Please follow this guide if you are new to CodeSmith Generator and are trialing the software.

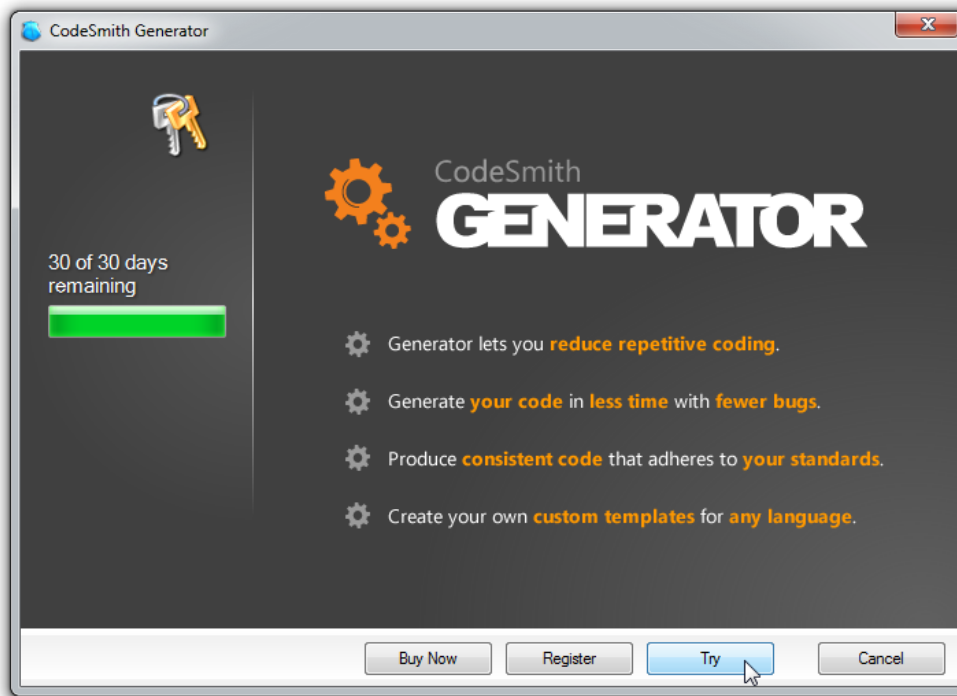


 If you need a Trial Key, you can click on the Request Trial Key link on the left column of this dialog to request a Trial Key.

 Please ensure that the Trial Key you enter in doesn't have any punctuation or trailing or leading spaces before pressing the Try Button.

If you are using a Trial version of CodeSmith Generator you need to enter in your trial key and click the Try button. **If you already own a license you must click the register button and follow the steps defined below.**

If you proceed in trial mode, CodeSmith Generator will display the Trial Mode dialog box:

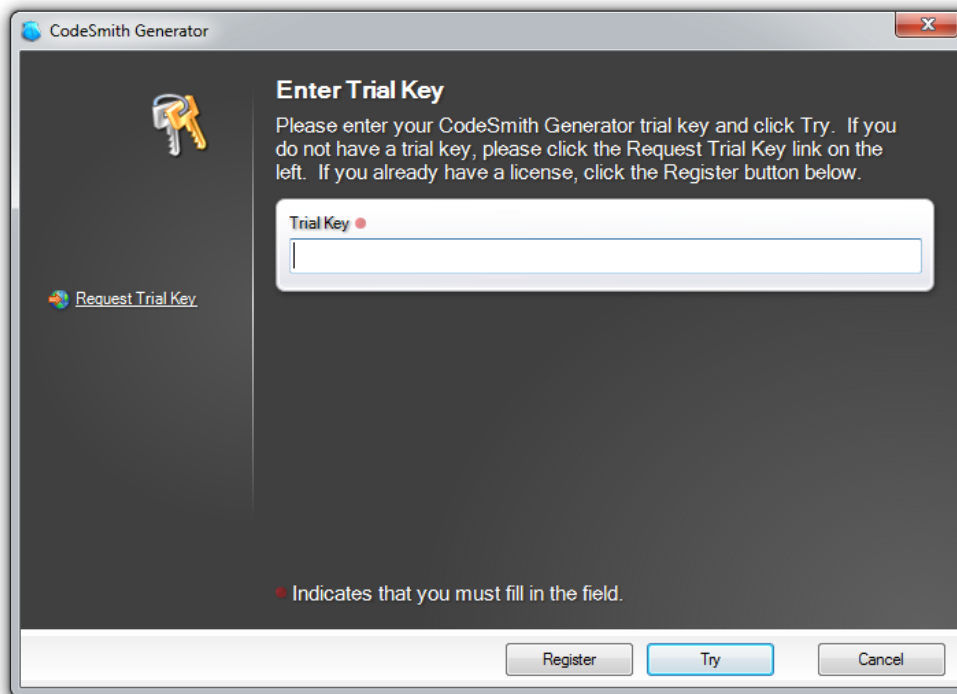


From this dialog box, you may:

- Click **Register** to return to the Enter Registration Information dialog box.
- Click **Buy Now** to learn more about purchasing CodeSmith Generator.
- Click **Try** to proceed with your CodeSmith Generator session.
- Click **Cancel** to abort your CodeSmith Generator session.

### ***Non-Trial Activation***


Please follow these steps if you have already purchased CodeSmith Generator.




From the first screen, you will want to select the **Register** button.

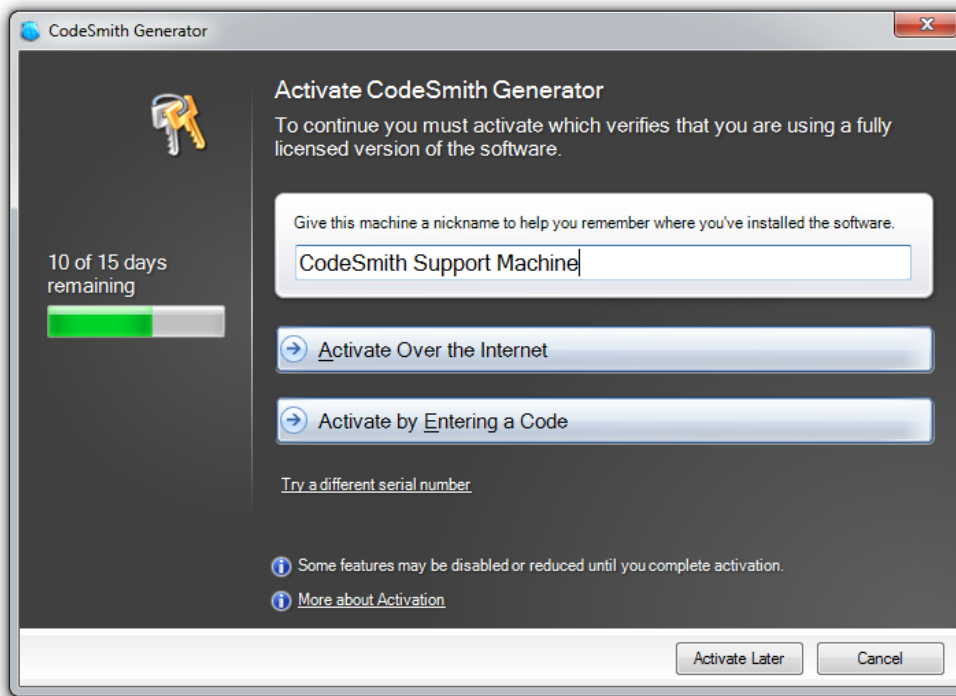


Please enter your name, organization information, email address, and the serial number that you received at the time of purchase. Click the **Register** button to continue with the activation process. If activation is successful, this activation dialog will close.

 Please contact Sales if you have lost your serial number. Please provide an Order Number if possible.

 Please ensure that the Serial Key you enter in doesn't have any punctuation or trailing or leading spaces before pressing the Register Button.

When you enter registration information with a valid serial number, CodeSmith Generator will display the following Activation Required dialog box if CodeSmith Generator cannot connect to the internet:

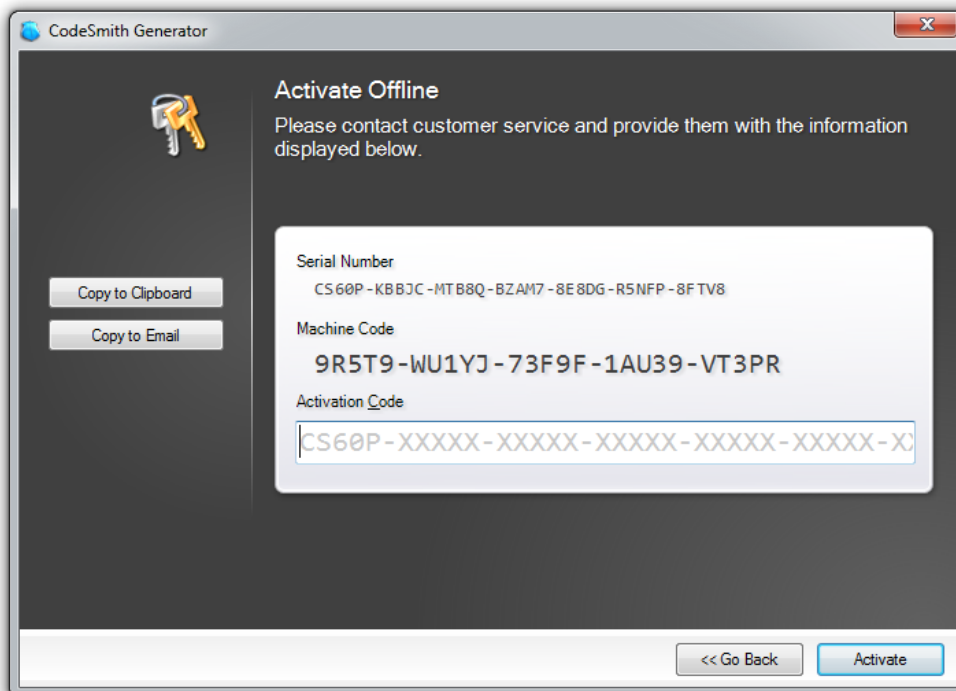


At this point, you have four choices. You may:

- Select **Activate Over the Internet** if your computer is connected to the Internet. This will attempt to automatically contact the CodeSmith licensing servers to activate this installation without further intervention on your part. If your computer accesses the Internet through a proxy server, click the Proxy Info link to enter your proxy server address, user name, and password.
- Select **Activate by Entering a Code** to manually enter activation information. This option will allow you to activate by phone or Email.
- Select the **Activate Later** button to continue without activation. This will postpone activation to a later time.
- Select the **Cancel** button to cancel the activation process.

Click on one of the four options to proceed.


If you choose the option to activate by Email or phone, CodeSmith Generator will display your serial number and machine key. *Please note that the machine key is randomly generated by CodeSmith.*




Click the **Copy to Clipboard** to copy the information from this dialog to paste in an email. Please send an e-mail message with the above information to [support@codesmithtools.com](mailto:support@codesmithtools.com).

You can also click the **Copy to Email** to send us an Email pre-populated with the above information.

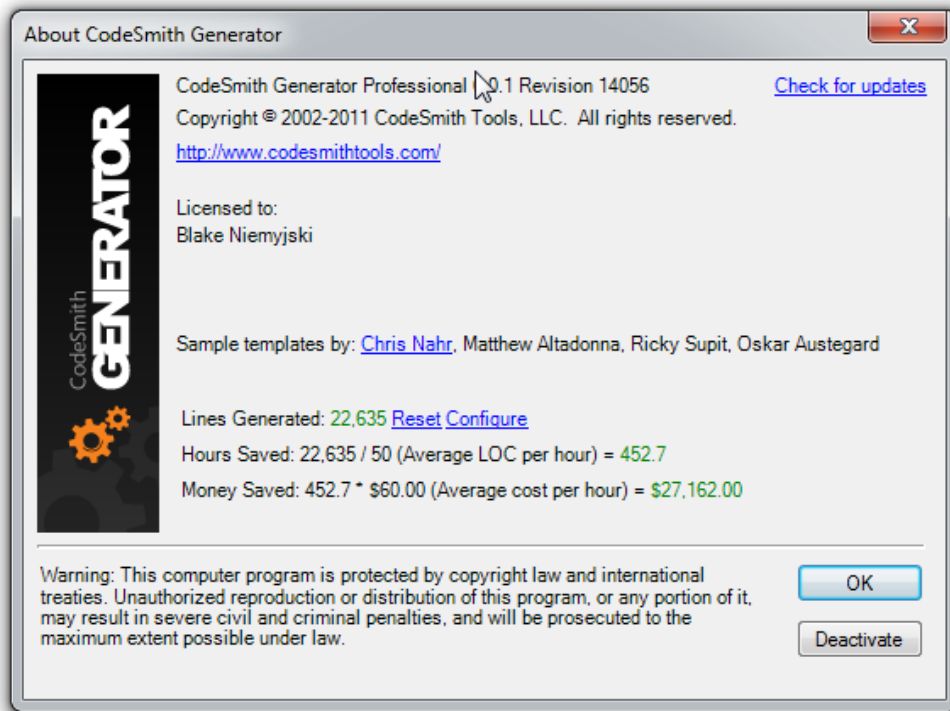
You will receive your unlock code by return e-mail. Paste the unlock code into this dialog box and click Continue to finish the activation process.

 The serial number and machine key shown are for illustration only. Be sure to use the information from your own computer when contacting support.

 Please ensure that the Activation Code you enter in doesn't have any punctuation or trailing or leading spaces before pressing the Activate Button.

## Deactivation

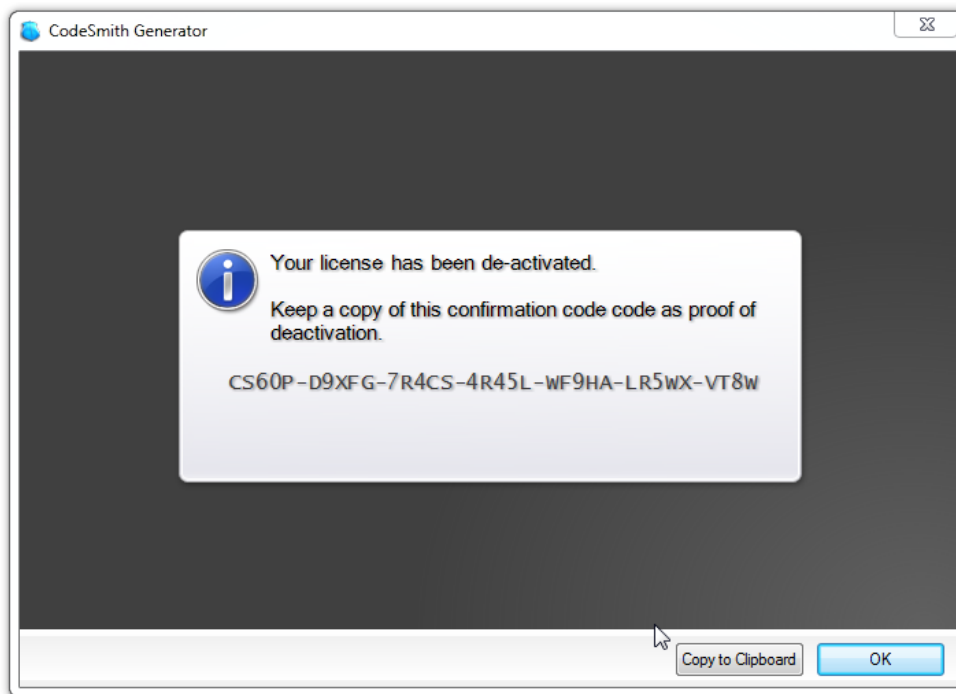
To deactivate your license you need to open the About CodeSmith Generator dialog.




Next, click on the **Deactivate** button to start the deactivation process.



To finish the deactivation process click on the **Deactivate** button and select **Yes** to the following dialog.



 Please note that the following generated key is for your records ONLY. To activate CodeSmith Generator on a new machine please use your original serial number.

### Activation Support

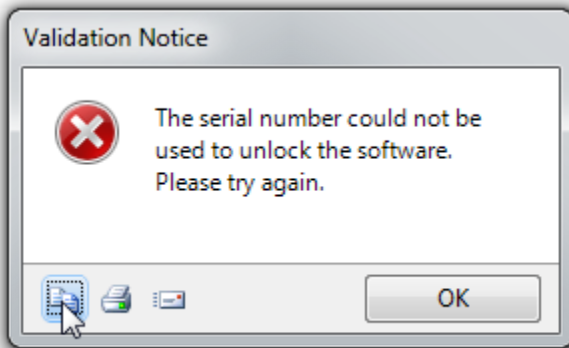
We strive for excellence, in the event that the activation process fails, please contact [support](#) as soon as possible with the following information.

1. The Serial Key you are trying to activate (E.G., CSX0P-XXXXX...),



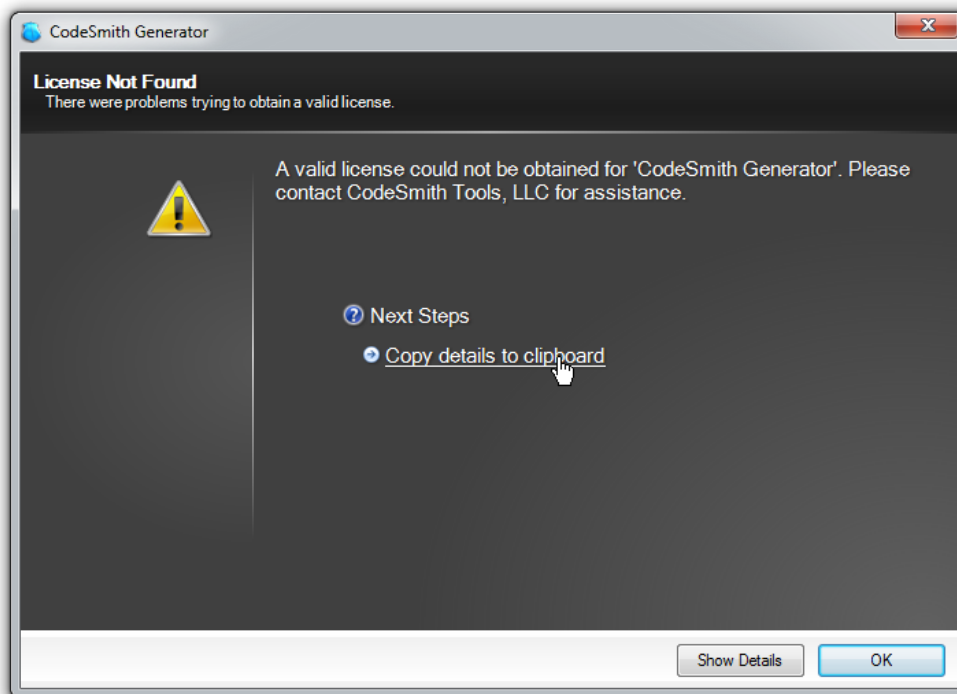
2. Operating System Information (E.G., Windows 7 SP1 64bit),
3. The exact version of CodeSmith Generator that you installed (E.G., X.X.X.XXXXX)
4. **A copy of the Activation Error information (Shown Below)**

Also, if you receive a Validation Notice dialog box while trying to activate CodeSmith Generator like the one below:



Please click on the lower left icon where the mouse icon is located. Doing so will copy any information available for the current dialog to the clipboard. Please also attach this information that is in the clipboard along with the information outlined in steps one through three.

**Next, please follow the step below.**



Please continue pressing the **Cancel** buttons on each dialog that you see until you get to the following screen.

Once you see this dialog, please click on the '**Copy details to clipboard**' link to copy this information to the clipboard. Finally attach this information as done in the previous step and send us the email.